JAVA / J2EE

By

Mr. Ashok

Program: A program is a set of instructions, which are executed by a machine to reduce the burden of a user or a human being by performing the operations faster without any mistakes.

Software: It is a set of programs, which can perform multiple tasks.

The softwares are classified into two types**:**

1) **System software:** The softwares which are designed to interact or communicate with the hardware devices, to make them work are called as system software. These softwares are generally developed in languages like C, C++ etc.

   Example: Operating System, drivers, compilers etc.

2) **Application Software:** The softwares which are called designed to store data provide entertainment, process data, do business, generate reports etc. are called as application software. These softwares are generally developed in languages like java, .net, etc.
   The application softwares are further classified into two types.
   i. **Stand Alone Software:** The software which can execute in the context of a single machine are called as standalone software.
      Example: MS-Word, media player, etc.
   ii. **Web Based Software:** This software can execute on any machine in the context of a browser are called as web based software.
      Example: Gmail, Facebook, etc

What is Java?

Java is a high level programming language and also known as platform because of its JRE (java runtime environment).

Brief History of Java

Java is one of the world's most important and widely used computer languages, and it has held this distinction for many years. Unlike some other computer languages whose influence has weared with passage of time, while Java's has grown.

As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers using and working on it.

Java language project initially started in June 1991 by James Gosling, Mike Sheridan, and Patrick Naughton. An oak tree stood outside Gosling's office at that time and java named as oak initially. It later renamed as Green and was later renamed as java from java coffee.

Java released versions:

JDK Alpha and Beta (1995)
JDK 1.0 (23rd Jan, 1996)
JDK 1.1 (19th Feb, 1997)
J2SE 1.2 (8th Dec, 1998)
J2SE 1.3 (8th May, 2000)
J2SE 1.4 (6th Feb, 2002)
J2SE 5.0 (30th Sep, 2004)
Java SE 6 (11th Dec, 2006)
Java SE 7 (28th July, 2011)
Java SE 8 (18th March, 2014)

As per the sun micro system standard the JAVA language is divided into three Editions.

1. **J2SE/JSE**
2. **J2EE/JEE**
3. **J2ME/JME**

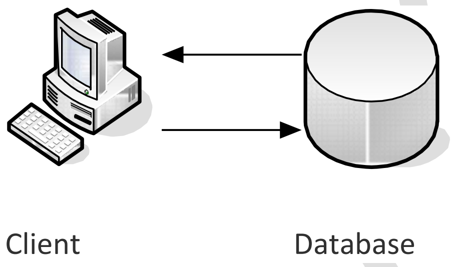
J2SE

- > J2SE stands for java 2 standard edition
- > By using J2SE we can develop standalone/Desktop applications.

Ex: notepad, WordPad, paint, Eclipse IDE………..etc.

Standalone applications: - Any application which runs in single computer is called standalone application.

i. Standalone applications are the java applications which don't need the client server architecture.
ii. The standalone applications applicable for the only one desktop hence it is called desktop applications or window based applications.
iii. For the standalone applications doesn't need internet connections.
iv. It is a local application it doesn't need any other external application support.
v. This type of the applications we can launch by using the command line or by using the executable jar.



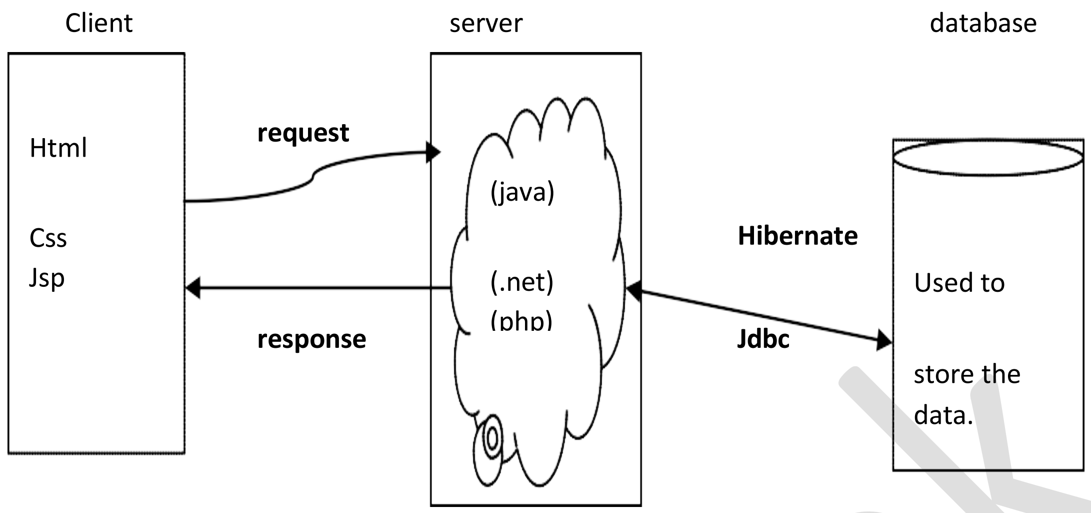
J2EE

-> J2EE stands for java 2 enterprise edition
-> By using J2EE we can develop web based applications.

Ex: Facebook, IRCTC, Flipkart etc.…

Web-applications:- Any application which can be accessible through the browser is called Web application

i. Web applications are the java applications which needs client and server concept.
ii. Web applications must need the internet connections to access the application.
iii. The application which is present in the internet is called the web application.



Client: The Person who is sending the request is called client. All browsers comes under Clients.
Ex: Google chrome, Mozilla Firefox, Internet Explorer etc.…

Server: In information technology, a server is a computer program that provides services to other computer programs (and their clients) in the same or other computers.
Ex: Apache Tomcat, Oracle WebLogic, IBM WebSphere's (WAS) etc.….

Database: Database is used to store the details like client details, application details, registration details……etc.
Ex: Oracle, DB2, MySQL etc.….

J2ME

- > J2ME stands for java 2 mobile edition
- > J2ME is used to develop mobile/micro based applications.

Ex: WhatsApp, mynthra etc...

Why java is used?

The prime reason behind creation of Java was to bring platform-independency, portability and security feature into a computer language. Beside these two major features, there were many other features that played an important role in moulding out the final form of this outstanding language. Those features are given below

Java features:

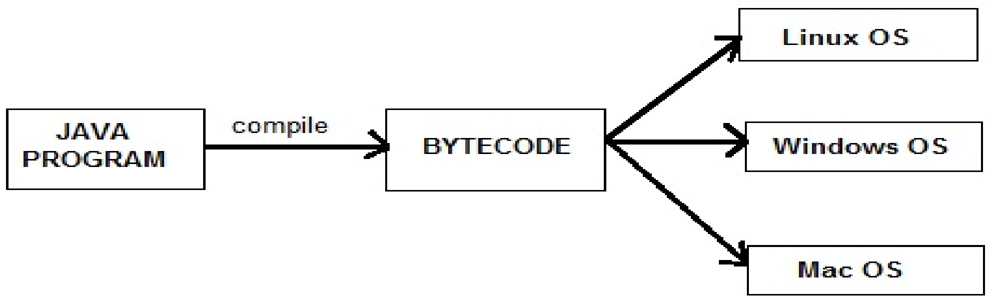
1. Simple, easy and familiar:

Java is easy to learn and familiar because
a) The syntaxes of java language are very similar with C and C++ syntaxes
b) Eliminated the complex topics of C and C++ in Java, those are pointers and operator overloading.
Because of these factors Java is very easy to learn.

2. Platform Independent:

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.
On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on

any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



3. Object-Oriented:

In java everything is Object which has some data and behaviour. Java can be easily extended as it is based on Object Model.

4. Robust:

Any technology if it is good at two main areas it is said to be ROBUST

i. Exception Handling
ii. Memory Allocation

JAVA is Robust because

JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.

JAVA is having very good memory management system that is Dynamic Memory (at runtime the memory is allocated) Allocation which allocates and deallocates memory for objects at runtime.

JVM – It will take care of allocating the memory
Garbage Collector – It will take care of de-allocating the memory

5. Secure:

When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

6. Distributed:

Java provides the network facility. I.e. programs can be access remotely from any machine on the network rather than writing program on the local machine. HTTP and FTP protocols are developed in java.

7. Portable:

Means able to be easily carried or moved. Write once, run anywhere (WORA) feature makes it portable.

8. Architecture-Neutral:

Java code is translated into byte code after compilation which is independent of any computer architecture, it needs only JVM (Java Virtual Machine) to execute.

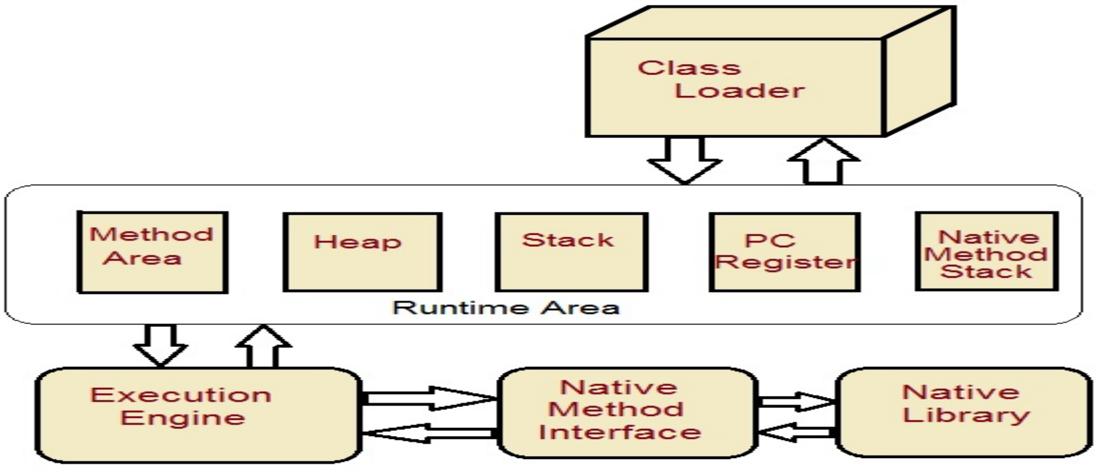
9. High performance:

JVM can execute byte codes (highly optimized) very fast with the help of Just in time (JIT) compilation technique.

10. Multithreading:
Java provides multitasking facility with the help of lightweight processes called threads.
11. Dynamic:
Java have the capability of linking dynamic new classes, methods and objects.

What is JVM?

Java virtual Machine (JVM) is a virtual Machine that provides runtime environment to execute java byte code. The JVM doesn't understand Java typo, that's why you compile your *.java files to obtain *.class files that contain the bytecodes understandable by the JVM.
JVM control execution of every Java program. It enables features such as automated exception handling, Garbage-collected heap.



JVM details:

1. Class loader subsystem:
It is a part of JVM that care of finding and loading of class files.
2. Class/method area:
It is a part of JVM that contains the information of types/classes loaded by class loader. It also contain the static variable, method body etc.

3. Heap:
It is a part of JVM that contains object. When a new object is created, memory is allocated to the object from the heap and object is no longer referenced memory is reclaimed by garbage collector.

4. Java Stack:
It is a part of JVM that contains local variable, operands and frames. To perform an operation, Byte code instructions takes operands from the stack, operate and then return the result in to the java stack.

5. Program Counter:
For each thread JVM instance provide a separate program counter (PC) or pc register which contains the address of currently executed instruction.

6. Native method stack:
As java program can call native methods (A method written in other language like c). Native method stack contains these native methods.

7. Execution Engine:

It is a part JVM that uses Virtual processor (for execution), interpreter (for reading instructions) and JIT (Just in time) compiler (for performance improvement) to execute the instructions.

How JVM is created (Why JVM is virtual):

When JRE installed on your machine, you got all required code to create JVM. JVM is created when you run a java program, e.g. If you create a java program named FirstJavaProgram.java. To compile use – java FirstJavaProgram.java and to execute use – java FirstJavaProgram. When you run second command – java FirstJavaProgram, JVM is created. That's why it is virtual.

Lifetime of JVM: When an application starts, a runtime instance is created. When application ends, runtime environment destroyed. If n no. of applications starts on one machine then n no. of runtime instances are created and every application run on its own JVM instance.

Main task of JVM:

i. Search and locate the required files.
ii. Convert byte code into executable code.
iii. Allocate the memory into ram
iv. Execute the code.
v. Delete the executable code.

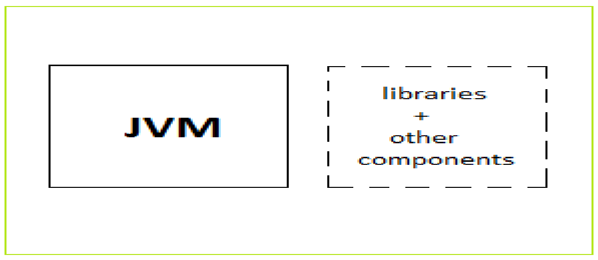
Difference between JVM, JRE and JDK

JVM (Java Virtual Machine):

JVM is a virtual machine or a program that provides run-time environment in which java byte code can be executed. JVMs are available for many hardware and software platforms. The use of the same byte code for all JVMs on all platforms make java platform independent.

JRE (Java Runtime Environment):

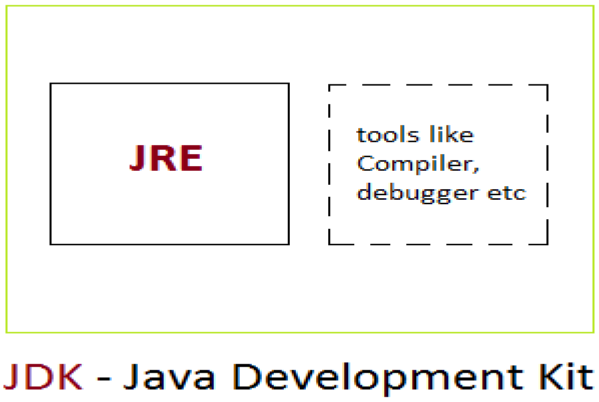
JVM + java runtime libraries + java package classes (e.g. util, Lang etc.). JRE provides class libraries and other supporting files with JVM. It not provide any development tool like compiler, debugger etc.



JRE - Java Runtime Environment

JDK (Java Development Kit):

JRE+ development tool (compiler, debugger etc.). JDK contains tools to develop the application and JRE to execute the application.



JAVA ENVIROMENT SETUP

Install the software and set the path:-

Download the software from internet based on your operating system. The software is different from 32-bit operating and 64-bit operating system.

To download the software open the following web site.

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

For 32-bit operating system please click on

Windows x86 :- 32- bit operating system

For 64-bit operating system please click on

Windows x64 :- 64-bit operating system

After installing the software the java folder is available in the fallowing location

Local Disk c: ------program Files--------java----jdk (java development kit), jre (java runtime environment)

To check whether the java is installed in your system or not go to the command prompt. To open the command prompt

In the command prompt type: - javac (Click on Enter)

'javac' is not recognized is an internal or external command, operable program or batch file.

Whenever we are getting above information at that moment the java is installed but the java is not working properly.

Whenever we are typing javac command on the command prompt

1. Operating system will pick up javac command search it in the internal operating system calls. The javac not available in the internal command list.
2. Then operating system goes to environmental variables and check is there any path is sets or not. Up to now we are not setting any path. So operating system don't know anything about javac command Because of this reason we are getting error message.

Hence we have to set environmental variables. The main aim of the setting environmental variable is to make available the following commands javac,java,javap (softwares) to the operating system.

To set the environmental variable:-

My Computer (right click on that) ---->properties----->Advanced--->Environment Variables---->
User variables > click on new button and enter below values

Variable name : **Path**
Variable value: **C:\Program Files (x86)\Java\jdk1.7.0_79\bin;.;**

Click on Ok -> Ok ->

Now the java is working fine in your system. Open the command prompt to check once C:>javac---------
□now list of commands will be displayed

If List of commands displayed on Console, that means java installed successfully.

Java Coding Conventions (Hungarian Notations)

Every predefined class, interface, method etc. will follow the java coding conventions and we are recommended to follow the same conventions. It is always a good programming practice to develop a program by following the coding conventions.

1. **Conventions for a class:** A class name can contain any number of words and the first letter of every word should be specified in uppercase.
   Example: System StringBuffer SecondProgram
2. **Convention for an interface:** An interface name can contain any number of words and the first letter of every word should be specified in uppercase.
   Example: Runnable ActionListener MyInterface
3. **Convention for a method:** A method name can contain any number of words and the first word should be specified completely in lowercase. The first letter of the remaining words if available should be specified in uppercase.
   Example: main() toCharArray() areaOfSquare()
4. **Convention for a variable:** A variable name can contain any number of words and the first word should be specified completely in lowercase. The first letter of the remaining words, if available should be specified in uppercase.
   Example: length numberOfStudents accountNumber
5. **Convention for a constant:** A constant can contain any number of words. All the letters of all the words should be specified in uppercase, if multiple words are available then separate them by underscore (_).
   Example: MIN_VALUE MAX_PRIORITY PI
6. **Convention for a package:** A package name should be specified completely in lowercase.
   Example: java.lang java.net ashoksoft.com

What is Variable?

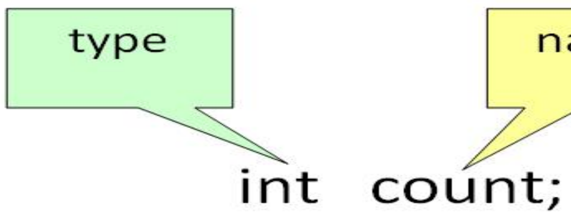
A variable can be thought of as a container which holds value for you during the life of your program. Every variable is assigned a **data type** which designates the type and quantity of a value it can hold.

Variable in program need to perform 2 steps

1. Variable Declaration
2. Variable Initialization

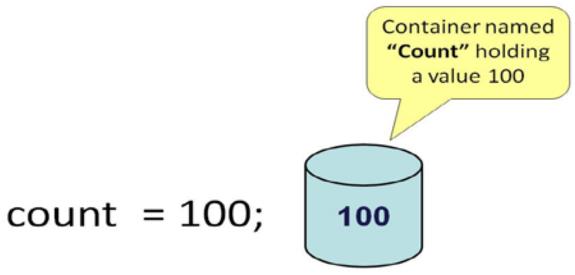
1) Variable Declaration:

To declare a variable, you must specify the data type & give the variable a unique name.



2) Variable Initialization:

To initialize a variable you must assign it a valid value.



Variables in Java

Java Programming language defines mainly three kind of variables, they are

- Instance variables
- Static Variables
- Local Variables

Instance variables

Instance variables are variables that are declare inside a class but outside any method, constructor or block. Instance variable are also variable of object commonly known as field or property.

```
class Student
{
 String name;
 int age;
}
```

Here name and age are instance variable of Student class.

Rules for Instance variable

- ✓ Instance variables can use any of the four access levels
- ✓ They can be marked final
- ✓ They can be marked transient
- ✓ They cannot be marked abstract
- ✓ They cannot be marked synchronized
- ✓ They cannot be marked strictfp
- ✓ They cannot be marked native
- ✓ They cannot be marked static

Static variables

Static are class variables declared with static keyword. Static variables are initialized only once. Static variables are also used in declaring constant along with final keyword.

```
class Student
{
 String name;
 int age;
 static int instituteCode=1101;
}
```

Here instituteCode is a static variable. Each object of Student class will share instituteCode property.

What is Static?

Static is a Non Access Modifier.

Applicable to

The Static keyword can be applied to

- ✓ Method
- ✓ Variable
- ✓ Class nested within another Class
- ✓ Initialization Block

Not Applicable to

The Static keyword cannot be applied to

- Class (Not Nested)
- Constructor
- Interfaces
- Method Local Inner Class(Difference then nested class)
- Inner Class methods
- Instance Variables
- Local Variables

Purpose of Static Keyword in Java

The static word can be used to attach a Variable or Method to a Class. The variable or Method that are marked static belong to the Class rather than to any particular instance.

How to Invoke

Static variable and Methods can be used without having any instance of the Class. Only the Class is required to invoke a static Method or static variable.

Class Variables – Static Fields

Class variables also known as static fields share characteristics across all Objects within a Class. When you declare a field to be static, only a single instance of the associated variable is created, which is common to all the Objects of that Class. Hence when one Object changes the value of a Class variable, it affects all the Objects of the Class. We can access a Class variable by using the name of the Class, and not necessarily using a reference to an individual Object within the Class. Static variables can be accessed even when no Objects of that Class exists. Class variables are declared using the static keyword.

Class Methods – Static Methods

Class Methods, similar to Class variables can be invoked without having an instance of the Class. Class methods are often used to provide global functions for Java programs. For example, Methods in the java.lang.Math package are Class methods. You cannot call non-static Methods from inside a static Method.

Static Keyword Rules

Variable or Methods marked static belong to the Class rather than to any particular Instance.
Static Method or variable can be used without creating or referencing any instance of the Class.
If there are instances, a static variable of a Class will be shared by all instances of that class, this will result in only one copy.
A static Method can't access a non-static variable nor can directly invoke non static Method (It can invoke or access Method or variable via instances).

Local variables

Local variables are declared in method constructor or blocks. Local variables are initialized when method or constructor block start and will be destroyed once its end. Local variable reside in stack. Access modifiers are not used for local variable.

```
float getDiscount(int price)
{
 float discount;
 discount=price*(20/100);
 return discount;
}
```

Here discount is a local variable.

Rule for Local Variable

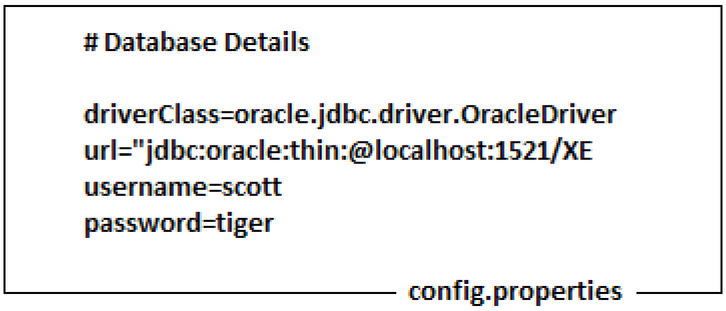
- Local variables cannot use any of the access level since its scope is only inside the method.
- Final is the Only Non-Access Modifier that can be applied to a local variable.
- Local variables are not assigned a default value, hence they need to be initialized.

Working with Properties files in java

Today, any complex application will require some sort of configuration. Sometimes we need this configuration to be read-only (mostly read at time of application startup) and sometimes (or rarely) we need to write back or update something on these property configuration files.

Properties class in Java

The **java.util.Properties** class provides API for reading and writing properties in form of **key=value** pairs. The properties file will be in plain text (.properties) format. For example:



```
# Database Details

driverClass=oracle.jdbc.driver.OracleDriver
url="jdbc:oracle:thin:@localhost:1521/XE
username=scott
password=tiger
```

config.properties

It can be used to get property value based on the property key. The Properties class provides methods to get data from properties file and store data into properties file. Moreover, it can be used to get properties of system.

Advantage of properties file

Recompilation is not required, if any information is changed from the properties file, you don't need to recompile the java class.

Reading Properties file

The following code loads **config.properties** file and read out a property called “username”:

```
public class ReadPropertiesDemo{
        public static void main(String... args){
        try {
                FileReader reader = new FileReader(configFile);
                Properties props = new Properties();
                props.load(reader);

                String uname = props.getProperty("username");

                System.out.print("Username name is: " + uname );

                reader.close();
        } catch (FileNotFoundException ex) {
                  // file does not exist
        } catch (IOException ex) {
            // I/O error
        }
    }
}
```

ReadPropertiesDemo.java

Writing Data to properties file

The following code writes value of a property to the config.properties file:

```
File configFile = new File("config.properties");

try {
   Properties props = new Properties();
   props.setProperty("host", "www.ashoksoft.com");
   FileWriter writer = new FileWriter(configFile);
   props.store(writer, "host settings");
   writer.close();
} catch (FileNotFoundException ex) {
   // file does not exist
} catch (IOException ex) {
   // I/O error
}
```

Understanding Class.forName (String className)

Dynamic loading of Java classes at runtime provides tremendous flexibility in the development of enterprise systems. It provides for the basis of “application servers”, and allows even simpler, lighter-weight systems to accomplish some of the same ends. Within Java, dynamic loading is typically achieved by calling the forName method on the class
java.lang.Class;

A call to Class.forName ("X") causes the class named X to be dynamically loaded (at runtime). A call to forName ("X") causes the class named X to be initialized (i.e., JVM executes all its static block after class loading). Class.forName

("X") returns the Class object associated with the "X" class. The returned Class object is not an instance of the "x" class itself.

Class.forName ("X") loads the class if it not already loaded. The JVM keeps track of all the classes that have been previously loaded. This method uses the classloader of the class that invokes it. The "X" is the fully qualified name of the desired class.

In general Class.forName is used to load the class dynamically where we doesn't know the class name beforehand. Once the class is loaded we will use newInstance () method to create the object dynamically. Let's consider that we have a class "Test", and we make a call like Class.forName("com.ashok.apps.Test"), then Test class will be initialized (JVM will run the static block which is inside the Test Class).Class.forName("com.ashok.apps.Test") will returns a Class object associated with class Test.

Let's see the below example, Our Test class will have a static block and a public constructor.

=======================================Test.java===

```
package com.ashok.apps;

public class Test
{
   static
   {
      System.out.println("Static block called");
   }
   public Test ()
   {
      System.out.println ("Inside Test class constructor");
   }
}
```

====================================Logic.java===

```
package com.ashok.apps;

import java.util.Scanner;

public class Logic
{
   public static void main(String args[])
   {
      try {
         String someClassName = "";
         Scanner in = new Scanner(System.in);
         System.out.print("Please class name with package structure");
         someClassName = in.nextLine();
         Class clasz = Class.forName(someClassName);
         Object obj = clasz.newInstance();
      }
      catch (ClassNotFoundException e)
      {
         e.printStackTrace();
      } catch (InstantiationException e)
      {
         e.printStackTrace();
      } catch (IllegalAccessException e)
```

```
                        {
                           e.printStackTrace();
                        }
                    }
                }
```
===0===

We may be in situations where in which you may know the class name beforehand, then we can use the above way to create object at runtime. Let’s see the explanation of the above code

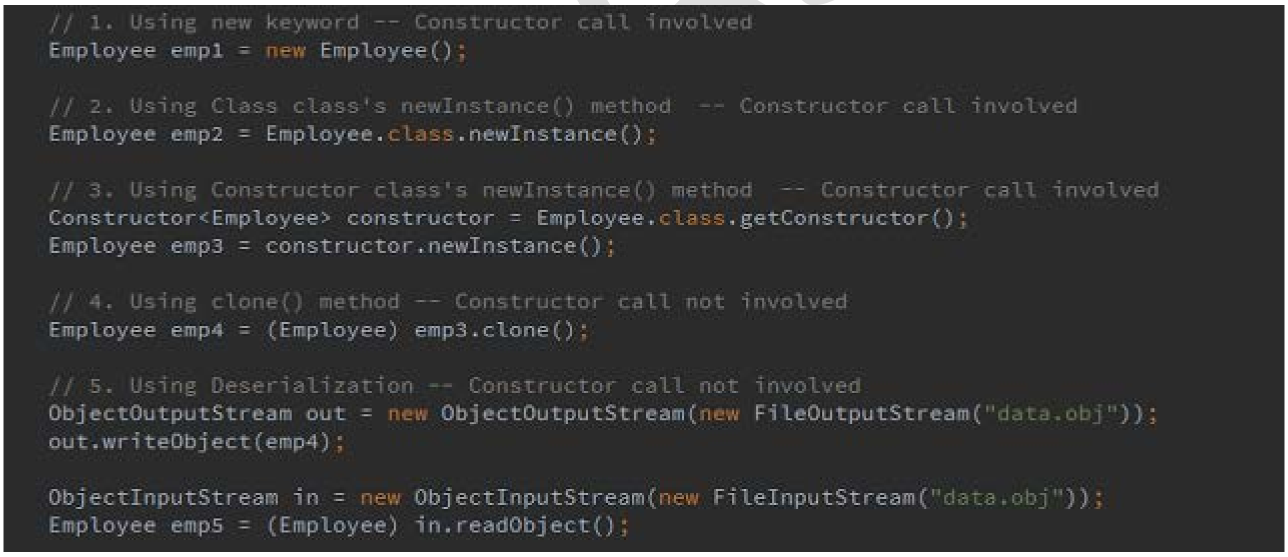
Through Scanner we will get the class name with full package structure entered in the console.

```
Scanner in = new Scanner (System.in);
System.out.println ("Please class name with package structure");
someClassName = in.nextLine ();
```
The below line creates the object of type Class which encapsulates the class provided by the user.
```
                    Class clasz = Class.forName(someClassName);
```
The class Class has a method newInstance() which will create object for the class entered by the user(Test)
```
                    Object obj = clasz.newInstance();
```
Finally we have created the object dynamically for a class without knowing its name beforehand.

How many ways are there to create Object for a class in java?

While being a Java developer we usually create lots of objects daily, but we always use the new or dependency management systems e.g. Spring to create these objects. However, there are more ways to create objects.

There are total 5 core ways to create objects in Java which are explained below


```
// 1. Using new keyword -- Constructor call involved
Employee emp1 = new Employee();

// 2. Using Class class's newInstance() method  -- Constructor call involved
Employee emp2 = Employee.class.newInstance();

// 3. Using Constructor class's newInstance() method  -- Constructor call involved
Constructor<Employee> constructor = Employee.class.getConstructor();
Employee emp3 = constructor.newInstance();

// 4. Using clone() method -- Constructor call not involved
Employee emp4 = (Employee) emp3.clone();

// 5. Using Deserialization -- Constructor call not involved
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("data.obj"));
out.writeObject(emp4);

ObjectInputStream in = new ObjectInputStream(new FileInputStream("data.obj"));
Employee emp5 = (Employee) in.readObject();
```

Approach 1. Using the new keyword

It is the most common and regular way to create an object and actually very simple one also. By using this method we can call whichever constructor we want to call (no-arg constructor as well as parametrised).

```
                    Employee emp1 = new Employee();
```

Approach 2. Using Class.newInstance() method

We can also use the newInstance() method of the Class class to create objects, This newInstance() method calls the no-arg constructor to create the object.

We can create objects by newInstance() in following way.

Employee emp2 = (Employee) Class.forName("com.ashok.apps.Employee").newInstance();

Or

Employee emp2 = Employee.class.newInstance();

Approach 3: Using newInstance() method of Constructor class

Similar to the newInstance() method of Class class, There is one newInstance() method in the java.lang.reflect.Constructor class which we can use to create objects. We can also call parameterized constructor, and private constructor by using this newInstance() method.

Both newInstance() methods are known as reflective ways to create objects. In fact newInstance() method of Class class internally uses newInstance() method of Constructor class. That's why the later one is preferred and also used by different frameworks like Spring, Hibernate, Struts etc.

```
Constructor<Employee> constructor = Employee.class.getConstructor();
Employee emp3 = constructor.newInstance();
```

Approach 4: Using clone () method

Whenever we call clone () on any object, JVM actually creates a new object for us and copy all content of the previous object into it. Creating an object using clone method does not invoke any constructor.

Note : To use clone () method on an object we need to implements Cloneable and define clone() method in it.

Employee emp4 = (Employee) emp3.clone ();

Java cloning is the most debatable topic in Java community and it surely does have its drawbacks but it is still the most popular and easy way of creating a copy of any object until that object is full filling mandatory conditions of Java cloning.

Approach 5. Using deserialization

Whenever we serialize and then deserialize an object, JVM creates a separate object for us. In deserialization, JVM doesn’t use any constructor to create the object.

Note: To Serialize & deserialize an object we need to implement the Serializable interface in our class.

```
ObjectInputStream in = new ObjectInputStream (new FileInputStream ("emp.ser"));
Employee emp5 = (Employee) in.readObject ();
```

Let’s consider an Employee class for which we are going to create the objects

==================================Employee.java===

```
package com.ashok;

import java.io.Serializable;

public class Employee implements Cloneable, Serializable {

	private static final long serialVersionUID = 1L;

	@Override
	public Object clone() {

		Object obj = null;
		try {
			obj = super.clone();
		} catch (CloneNotSupportedException e) {
			e.printStackTrace();
		}
		return obj;
	}
}
```

=====================================0==

In below java program we are going to create Employee objects in all 5 ways, you can also found the complete source code at Ashok IT School group in facebook.

==================================Test.java===

```
package com.ashok;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Constructor;

public class Test {
	public static void main(String[] args) throws Exception {

		// using new operator
		Employee emp1 = new Employee();
		System.out.print("Emp 1 - Hash Code : ");
		System.out.println(emp1.hashCode());

		// using Class.forName(.)
		Class c = Class.forName("com.ashok.Employee");
		Employee emp2 = (Employee) c.newInstance();
		System.out.print("Emp 2 - Hash Code : ");
		System.out.println(emp2.hashCode());

		// using newInstance()
		Constructor<Employee> con = Employee.class.getConstructor();
		Employee emp3 = con.newInstance();
		System.out.print("Emp 3 - Hash Code : ");
		System.out.println(emp3.hashCode());

		// using Cloning
		Employee emp4 = (Employee) emp3.clone();
		System.out.print("Emp 4 - Hash Code : ");
		System.out.println(emp4.hashCode());

		// Using De-Serialization
		FileOutputStream fos = new FileOutputStream("emp.ser");
		ObjectOutputStream oos = new ObjectOutputStream(fos);
		oos.writeObject(emp4);

		FileInputStream fis = new FileInputStream("emp.ser");
		ObjectInputStream ois = new ObjectInputStream(fis);
		Employee emp5 = (Employee) ois.readObject();
		System.out.print("Emp 5 - Hash Code : ");
		System.out.println(emp5.hashCode());
	}
```

}
===0==

Reflection in Java

In Java, the process of analyzing and modifying all the capabilities of a class at runtime is called Reflection.

Reflection API in Java is used to manipulate class and its members which include fields, methods, constructor, etc. at runtime.

Reflection in Java is a very powerful concept and it's of little use in normal programming but it's the backbone for most of the Java, J2EE frameworks. Some of the frameworks that use java reflection are:

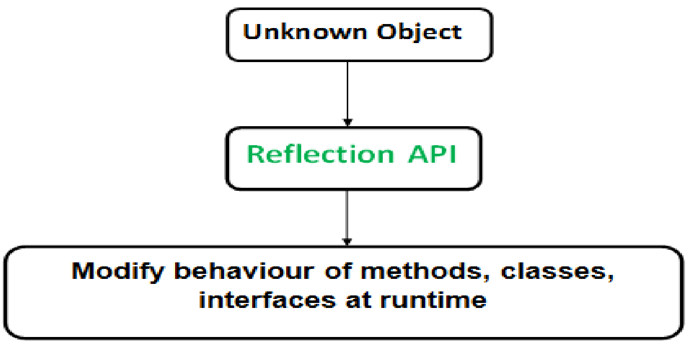
- **JUnit – uses reflection to parse @Test annotation to get the test methods and then invoke it.**
- **Spring – dependency injection, read more at Spring Dependency Injection**
- **Tomcat web container to forward the request to correct module by parsing their web.xml files and request URI.**
- **Eclipse auto completion of method names**
- **Struts**
- **Hibernate etc..**

The list is endless and they all use java reflection because all these frameworks have no knowledge and access of user defined classes, interfaces, their methods etc.

Reflection enables us to do below operations

✓ Examine an object's class at runtime
✓ Construct an object for a class at runtime
✓ Examine a class's field and method at runtime
✓ Invoke any method of an object at runtime
✓ Change accessibility flag of Constructor, Method and Field etc.

java.lang.Class is the entry point for all the reflection operations. For every type of object, JVM instantiates an immutable instance of java.lang.Class that provides methods to examine the runtime properties of the object and create new objects, invoke its method and get/set object fields.



The java.lang.reflect package provides many classes to use reflection, Following is a list of various Java classes in java.lang.package to implement reflection-

- **Field:** This class is used to gather declarative information such as datatype, access modifier, name and value of a variable.
- **Method:** This class is used to gather declarative information such as access modifier, return type, name, parameter types and exception type of a method.

- **Constructor:** This class is used to gather declarative information such as access modifier, name and parameter types of a constructor.
- **Modifier:** This class is used to gather information about a particular access modifier.
- **Public String getName ():** Returns the name of the class.
- **public Class getSuperclass():** Returns the super class reference
- **Public Class[] getInterfaces()** : Returns an array of interfaces implemented by the specified class
- **Public in getModifiers ():** Returns an integer value representing the modifiers of the specified class which needs to be passed as parameter to "public static String toString (int i)" method which returns the access specifier for the given class.

Example to demonstrate Reflection API

```
===================================Test.java=========================================
package com.ashoksoft;

// class whose object is to be created
public class Test {
	// creating a private field
	private String name;

	// creating a public constructor
	public Test() {
		name = "www.ashoksoft.com";
	}
	// Creating a public method with no arguments
	public void method1() {
		System.out.println("The name is " + name);
	}

	// Creating a public method with int as argument
	public void method2(int n) {
		System.out.println("The number is " + n);
	}

	// creating a private method
	private void method3() {
		System.out.println("Private method invoked");
	}
}
=====================Demo.java=================================
package com.ashoksoft;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

class Demo {
	public static void main(String args[]) throws Exception {
		// loading class into JVM
		Class cls = Class.forName("com.ashoksoft.Test");

		// Creating class object from the object using
		Test t = (Test) cls.newInstance();
```

```
System.out.println("The name of class is " + cls.getName());

// Getting the constructor of the class through the
// object of the class
Constructor constructor = cls.getConstructor();
System.out.println("The name of constructor is " + constructor.getName());

System.out.println("The public methods of class are : ");

// Getting methods of the class through the object
// of the class by using getMethods
Method[] methods = cls.getMethods();

// Printing method names
for (Method method : methods) {
        System.out.println(method.getName());
}

// creates object of desired method by providing the
// method name and parameter class as arguments to
// the getDeclaredMethod
Method methodcall1 = cls.getDeclaredMethod("method2", int.class);

// invokes the method at runtime
methodcall1.invoke(t, 19);

// creates object of the desired field by providing
// the name of field as argument to the
// getDeclaredField method
Field field = cls.getDeclaredField("name");

// allows the object to access the field irrespective
// of the access specifier used with the field
field.setAccessible(true);

// takes object and the new value to be assigned
// to the field as arguments
field.set(t, "JAVA");

// Creates object of desired method by providing the
// method name as argument to the getDeclaredMethod
Method methodcall2 = cls.getDeclaredMethod("method1");

// invokes the method at runtime
methodcall2.invoke(t);

// Creates object of the desired method by providing
// the name of method as argument to the
// getDeclaredMethod method
Method methodcall3 = cls.getDeclaredMethod("method3");

// allows the object to access the method irrespective
```

```
                // of the access specifier used with the method
                methodcall3.setAccessible(true);

                // invokes the method at runtime
                methodcall3.invoke(t);
        }
}
```

=============================0==

Summary:

- ➢ Reflection programming in java helps in retrieving and modifying information about Classes and Class members such variable, methods, constructors.
- ➢ Reflection API in Java can be implemented using classes in java.lang.reflect package and methods of java.lang.Class class.
- ➢ Some commonly used methods of java.lang.Class class are getName (), getSuperclass (), getInterfaces (), getModifiers () etc.
- ➢ Some commonly used classes in java.lang.reflect package are Field, Method, Constructor, Modifier, etc.
- ➢ Reflection API can access private methods and variables of a class which could be a security threat.
- ➢ Reflection API is a powerful capability provided by Java, but it comes with some overheads such as slower performance, security vulnerability, and permission issue. Hence, reflection API should be treated as the last resort to perform an operation.

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.
Recursion is a basic programming technique we can use in Java. One of the classic problems for introducing recursion is calculating the factorial of an integer. The factorial of any given integer — call it n so that you sound mathematical — is the product of all the integers from 1 to n. Thus, the factorial of 5 is 120: 5 x 4 x 3 x 2 x 1.
The recursive way to look at the factorial problem is to realize that the factorial for any given number n is equal to n times the factorial of n–1, provided that n is greater than 1. If n is 1, the factorial of n is 1.
This definition of factorial is recursive because the definition includes the factorial method itself. It also includes the most important part of any recursive method: an end condition. The end condition indicates when the recursive method should stop calling itself. In this case, when n is 1, it just returns 1. Without an end condition, the recursive method keeps calling itself forever.

Here's the recursive version of the factorial method:

---Factorial.java--

```
import java.util.Scanner;

public class Factorial {
  public static void main (String[ ] args) {
    System.out.println("Enter the number for factorial calculation : ");

    Scanner scanner = new Scanner(System.in);

    int a = scanner.nextInt();
    int result = fact(a);
    System.out.println("Factorial of " +a+ " is :"+result);
  }
```

```
        static int fact(int b){
         if(b <= 1)
           return 1;
         else
           return b * fact(b-1);
        }
       }
```

---0--

Requirement: Write a java program to reverse a String using recursion (without reverse () and loops)

Serialization and De-Serialization

Serialization is the mechanism of translating Java object's values and states to bytes to send it over network or to save it on file and disks. On other hand Deserialization is the process of converting byte code to corresponding java objects.

Serialization and Deserialization in Java

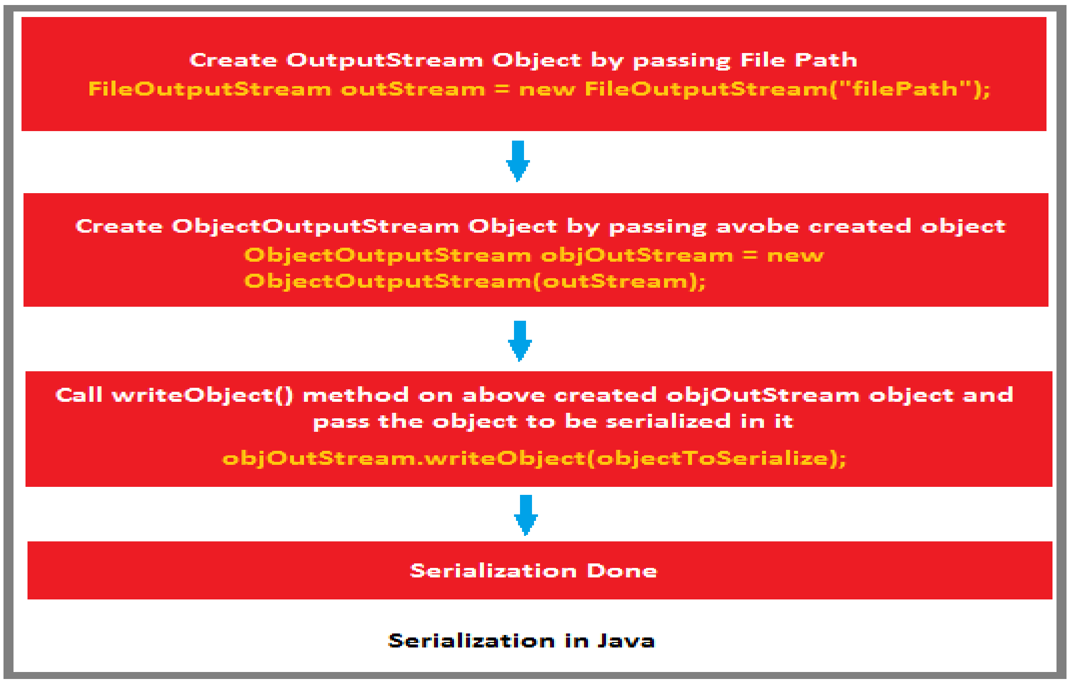
Java serialization API provides mechanism to write an object into streams, disks, db's or something…. so that the object can be transported over network or can be stored somewhere and can be rebuild again when needed. Getting serialized object again called deserialization.

Need of Serialization

Serialization comes into picture when we need to send an object(not text) over network or store in a file so that the object can be rebuild again when needed. But the network infrastructure and hard disks does not understand java, what they understand are bytes only, and this is what serialization does. It converts java object to bytes and bytes to java object again.

How to serialize an object in java

All steps are included in the image showing below, lets implement all those steps of serialization one by one to persist a java object to a specified file.


Create OutputStream Object by passing File Path
FileOutputStream outStream = new FileOutputStream("filePath");
Create ObjectOutputStream Object by passing avobe created object
ObjectOutputStream objOutStream = new ObjectOutputStream(outStream);
Call writeObject() method on above created objOutStream object and pass the object to be serialized in it
objOutStream.writeObject(objectToSerialize);
Serialization Done
Serialization in Java

Class to be serialized – Student.java

A class must implement Serializable interface in order to be serialized, this is a marker interface that does not contain any method in it.

```java
package com.ashoksoft;

import java.io.Serializable;

/**
 * @author Ashok Bollepalli
 *
 */
@SuppressWarnings("serial")
public class Student implements Serializable {

 private static final long serialVersionUID = 1L;

 private int id;
 private String name;

 public int getId() {
  return id;
 }

 public void setId(int id) {
  this.id = id;
 }

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

}
```

Serialization Implementation – SimpleSerImpl.java

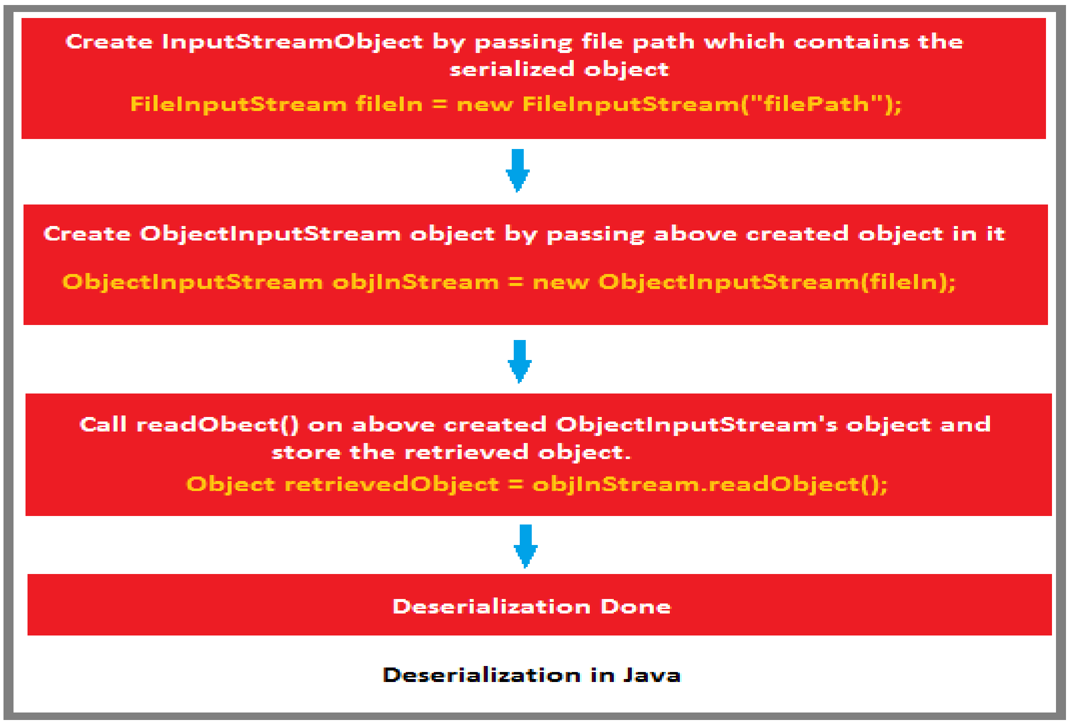
Here is code for serialization of java objects, we just need to get an FileOutputStream object from the five we want to persist the object and pass this to ObjectOutputStream's object's constructor. Method writeObject() is used to get bytes from java object and pass them to output stream.

```java
package com.ashoksoft;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
```

```java

/**
 * @author Ashok Bollepalli
 *
 */
public class SimpleSerImpl {
 public static void main(String args[]) {

  /*
   * Assigning values to Student classes
   */
  Student student = new Student();
  student.setId(1);
  student.setName("Ashok Bollepalli");

  /*
   * Serializing Student class
   */
  try {
   FileOutputStream fileOutputStream = new FileOutputStream(
     "serialobject.ser");
   ObjectOutputStream objectOutputStream = new ObjectOutputStream(
     fileOutputStream);
   objectOutputStream.writeObject(student);

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  }

 }
}
```

How to Deserialize an object in java

All steps are included in the image showing below, let’s implement all those steps of deserialization one by one to get java object back from the file.


Create InputStreamObject by passing file path which contains the serialized object
FileInputStream fileIn = new FileInputStream("filePath");
Create ObjectInputStream object by passing above created object in it
ObjectInputStream objInStream = new ObjectInputStream(fileIn);
Call readObect() on above created ObjectInputStream's object and store the retrieved object.
Object retrievedObject = objInStream.readObject();
Deserialization Done
Deserialization in Java

Deserialization Implementation – SimpleSerImpl.java

Here is code for deserialization of java objects, we just need to get an FileInputStream object from the file that contains object's bytes and pass this to ObjectInputStream's object's constructor. Method readObject() returns an object whose bytes are being read from the file.

```
package com.ashoksoft;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;

/**
 * @author Ashok Bollepalli
 *
 */
public class SimpleSerImpl {
 public static void main(String args[]) {

  /*
   * Deerializing Student class
   */
  Student studentOut = null;

  try {
   FileInputStream fileInputStream = new FileInputStream(
     "serialobject.ser");
   ObjectInputStream inputStream = new ObjectInputStream(
     fileInputStream);
   studentOut = (Student) inputStream.readObject();
  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  } catch (ClassNotFoundException cnf) {
   // TODO Auto-generated catch block
   cnf.printStackTrace();
  }

  /*
   * Printing values from deserialized object
   */
  System.out.println("Deserailization of Student's object :");
  System.out.println("ID: " + studentOut.getId());
  System.out.println("Name: " + studentOut.getName());

 }
}
```

Here we are done with Serialization and Deserialization of a simple object in java, but in real world applications many other situation may occur. Let's look a little deep in serialization in different scenarios.

What if the serialized object has a reference to another object

In case the object we are going to serialize has some reference to other object, then what would happen? In this case three possible conditions can arise.

CASE 1: If the referenced class implements Serializable
In case we have a reference to another object and the referenced class also implements Serializable, that the referenced class will be automatically serialized. In the example below, we have a Employee class that contains EmployeeAddress's reference in it, in this case the both classes are implementing Serializable interface so both will be serialized.

Class to be serialized - Employee.java

```
package com.ashoksoft;

import java.io.Serializable;
/**
 * @author Ashok Bollepalli
 *
 */
public class Employee implements Serializable {
 private int empId;
 private String empName;
 private EmployeeAddress employeeAddress;

 public int getEmpId() {
  return empId;
```

```java
 }

 public void setEmpId(int empId) {
  this.empId = empId;
 }

 public String getEmpName() {
  return empName;
 }

 public void setEmpName(String empName) {
  this.empName = empName;
 }

 public EmployeeAddress getEmployeeAddress() {
  return employeeAddress;
 }

 public void setEmployeeAddress(EmployeeAddress employeeAddress) {
  this.employeeAddress = employeeAddress;
 }

}
```

Referenced Class – EmployeeAddress

Here is a simple referenced class 'EmployeeAddress ' this class is being referenced by 'Employee' and implements serializable interface. This class w'll be serialized along with Employee.

```java
package com.ashoksoft;

import java.io.Serializable;
/**
 * @author Ashok Bollepalli
 *
 */
public class EmployeeAddress implements Serializable {

 private String street;
 private String city;
 private String state;
 private String country;

 public String getStreet() {
  return street;
 }
 public void setStreet(String street) {
  this.street = street;
 }
 public String getCity() {
  return city;
 }
 public void setCity(String city) {
  this.city = city;
 }
 public String getState() {
  return state;
 }
 public void setState(String state) {
  this.state = state;
 }
 public String getCountry() {
  return country;
 }
 public void setCountry(String country) {
  this.country = country;
 }

}
```

Serialization and De-serialization Implementation

Here is code for serialization and deserialization of java object with a reference to another serializable object. See the implementation below:

```
package com.ashoksoft;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
/**
 * @author Ashok
 *
 */
public class ReferenceSerImpl {
 public static void main(String args[]) {

  /*
   * Assigning values to Employee and EmployeeAdress classes
   */
  Employee emp = new Employee();
  emp.setEmpId(1);
  emp.setEmpName("Ashok Bollepalli");

  EmployeeAddress employeeAddress = new EmployeeAddress();
  employeeAddress.setStreet("Sec-44");
  employeeAddress.setCity("Delhi");
  employeeAddress.setState("Delhi");
  employeeAddress.setCountry("India");

  emp.setEmployeeAddress(employeeAddress);

  /*
   * Serializing Employee class, referenced class EmployeeAdress will be
   * serialized bydefault, unless the referenced class variable is not
   * declared transient
   */
  try {
   FileOutputStream fileOutputStream = new FileOutputStream(
     "serialObject.ser");
   ObjectOutputStream objectOutputStream = new ObjectOutputStream(
     fileOutputStream);
   objectOutputStream.writeObject(emp);

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  }

  /*
   * Deserializing Employee class, referenced class EmployeeAdress will be
   * deserialized bydefault, unless the referenced class variable is not
   * declared transient
   */
  Employee empout = null;

  try {
   FileInputStream fileInputStream = new FileInputStream(
     "serialobject.ser");
   ObjectInputStream inputStream = new ObjectInputStream(
     fileInputStream);
   empout = (Employee) inputStream.readObject();

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  } catch (ClassNotFoundException cnf) {
   // TODO Auto-generated catch block
   cnf.printStackTrace();
  }

  /*
   * Printing values from deserialized object
   */
  System.out.println("Deserailization of Employee and Address :");
  System.out.println("Emp ID: " + empout.getEmpId());
  System.out.println("Emp Name: " + empout.getEmpName());
  System.out.println("Emp Address Street: "
    + empout.getEmployeeAddress().getStreet());
```

```
System.out.println("Emp Address City: "
  + empout.getEmployeeAddress().getCity());
System.out.println("Emp Address State: "
  + empout.getEmployeeAddress().getState());
System.out.println("Emp Address Country: "
  + empout.getEmployeeAddress().getCountry());

 }
}
```

In case the reference class does not implement Serializable its variable must be declared transient.

CASE 2: If the referenced class does not implements Serializable but its reference variable is transient.
In case the referenced class EmployeeAddress does not implement Serializable interface, than there will be runtime error in serializing Employee class. To avoid the error just make EmployeeAddress's reference variable transient.

Transient EmployeeAddress employeeAddress; In this case when we will deserialize Employee class, there will be a null value for EmployeeAddress reference.

CASE 3: If the referenced class can not implement Serializable interface and we still want to persist its states.
In the above two cases we saw, if a reference variable is there either the referenced class must be serializable or the reference variable must be declared transient. Now the question arises, is it possible to persist states of a referenced class even if the class is not serializable and its reference variable is declared transient. Yes, This is possible !

In that case, Java serialization provides a mechnism such that if you have private methods with particular signature then they will get called during serialization and deserialization so we will override writeObject and readObject method of Employee class and they will be called during serialization and deserialization of Employee object.

Object To Serialize – Employee.java

```
package com.ashoksoft;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
/**
 * @author Ashok
 *
 */
public class Employee implements Serializable {
 private static final long serialVersionUID = 1L;
 private int empId;
 private String empName;
 transient private EmployeeAddress employeeAddress;

 public int getEmpId() {
  return empId;
```

```
}

public void setEmpId(int empId) {
 this.empId = empId;
}

public String getEmpName() {
 return empName;
}

public void setEmpName(String empName) {
 this.empName = empName;
}

public EmployeeAddress getEmployeeAddress() {
 return employeeAddress;
}

public void setEmployeeAddress(EmployeeAddress employeeAddress) {
 this.employeeAddress = employeeAddress;
}

private void writeObject(ObjectOutputStream ous) throws IOException,
  ClassNotFoundException {
 try {
  ous.defaultWriteObject();
  ous.writeObject(employeeAddress.getStreet());
  ous.writeObject(employeeAddress.getCity());
  ous.writeObject(employeeAddress.getState());
  ous.writeObject(employeeAddress.getCountry());

 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }
}

private void readObject(ObjectInputStream ois) throws IOException,
  ClassNotFoundException {
 try {
  ois.defaultReadObject();
  employeeAddress = new EmployeeAddress((String) ois.readObject(),
    (String) ois.readObject(), (String) ois.readObject(),
    (String) ois.readObject());
 } catch (ClassNotFoundException e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }

 }

}
```

Reference Object without Implementing Serializable – EmployeeAddress.java

Here is a simple referenced class 'EmployeeAddress ' this class is being referenced by 'Employee' and does not implements serializable interface.

```
package com.ashoksoft;

/**
 * @author Ashok Bollepalli
 *
 */
public class EmployeeAddress{

 private String street;
 private String city;
 private String state;
 private String country;

 EmployeeAddress(String street, String city, String state, String country){
  super();
  this.street = street;
  this.city = city;
  this.state = state;
  this.country = country;
 }

 public String getStreet() {
  return street;
 }
 public void setStreet(String street) {
  this.street = street;
 }
 public String getCity() {
  return city;
 }
 public void setCity(String city) {
  this.city = city;
 }
 public String getState() {
  return state;
 }
 public void setState(String state) {
  this.state = state;
 }
 public String getCountry() {
  return country;
 }
 public void setCountry(String country) {
  this.country = country;
```

```java
 }

}
```

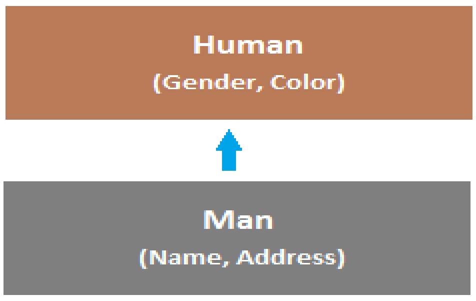
Implementation – Serialization And Deserialization

Here is code for serialization and deserialization of java object with a reference to another non serializable object. See the implementation below:

```java
package com.ashoksoft;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
/**
 * @author Ashok Bollepalli
 *
 */
public class ReferenceSerImpl {
 public static void main(String args[]) {

  /*
   * Assigning values to Employee and EmployeeAdress classes
   */
  Employee emp = new Employee();
  emp.setEmpId(1);
  emp.setEmpName("Nagesh Chauhan");

  EmployeeAddress employeeAddress = new EmployeeAddress("Sec-44","Delhi","Delhi","India");
//  employeeAddress.setStreet("Sec-44");
//  employeeAddress.setCity("Delhi");
//  employeeAddress.setState("Delhi");
//  employeeAddress.setCountry("India");
//
  emp.setEmployeeAddress(employeeAddress);

  /*
   * Serializing Employee class, referenced class EmployeeAdress will be
   * serialized bydefault, unless the referenced class variable is not
   * declared transient
   */
  try {
   FileOutputStream fileOutputStream = new FileOutputStream(
     "serialObject.ser");
   ObjectOutputStream objectOutputStream = new ObjectOutputStream(
     fileOutputStream);
   objectOutputStream.writeObject(emp);

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  }

  /*
   * Deserializing Employee class, referenced class EmployeeAdress will be
   * deserialized bydefault, unless the referenced class variable is not
   * declared transient
   */
  Employee empout = null;

  try {
   FileInputStream fileInputStream = new FileInputStream(
     "serialobject.ser");
   ObjectInputStream inputStream = new ObjectInputStream(
     fileInputStream);
   empout = (Employee) inputStream.readObject();

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  } catch (ClassNotFoundException cnf) {
   // TODO Auto-generated catch block
   cnf.printStackTrace();
  }

  /*
   * Printing values from deserialized object
   */
  System.out.println("Deserailization of Employee and Address :");
  System.out.println("Emp ID: " + empout.getEmpId());
  System.out.println("Emp Name: " + empout.getEmpName());
  System.out.println("Emp Address Street: "
    + empout.getEmployeeAddress().getStreet());
  System.out.println("Emp Address City: "
    + empout.getEmployeeAddress().getCity());
  System.out.println("Emp Address State: "
    + empout.getEmployeeAddress().getState());
  System.out.println("Emp Address Country: "
    + empout.getEmployeeAddress().getCountry());

 }
}
```

Inheritance in Java Serialization - Example and Explanation

In our previous discussions we came across, Serialization, serialVersionUID and transient variable. In this particular blog we will see 'Inheritance in Serialization'.



Inheritance in Java Serilization
(www.ashoksoft.com)

In case of inheritance, when we want to serialize an object there may be three possible scenarios.

(1) If Super class is Serializable

In case super class is Serializable than all its subclasses will be serializable by default. No need to implement serializable interface in subclass explicitly. See the implementation below.

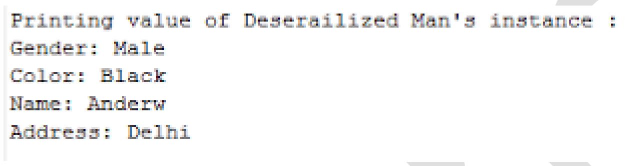
```java
package com.ashoksoft;

import java.io.Serializable;

/**
 * @author Ashok Bollepalli
 */
public class Human implements Serializable {

private static final long serialVersionUID = 1L;

String gender;
String color;

Human(String gender, String color) {
 this.gender = gender;
 this.color = color;
}

public String getGender() {
 return gender;
}

public void setGender(String gender) {
 this.gender = gender;
}

public String getColor() {
  return color;
 }

 public void setColor(String color) {
  this.color = color;
 }

}
```

```
package com.ashoksoft;

/**
 * @author Ashok Bollepalli
 */
public class Man extends Human {

 private String name;
 private String address;

 Man(String gender, String color, String name, String address) {
  super(gender, color);
  this.name = name;
  this.address = address;
 }

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public String getAddress() {
  return address;
 }

 public void setAddress(String address) {
  this.address = address;
 }

}
```

```
package com.ashoksoft;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

```java

/**
 * @author Ashok Bollepalli
 */
public class Implementation {
 public static void main(String args[]) {

  /*
   * Assigning values to Man class's instance
   */
  Man man = new Man("Male", "Black", "Anderw", "Delhi");

  /*
   * Serializing Man's instance
   */
  try {
   FileOutputStream fileOutputStream = new FileOutputStream(
     "serialObject.ser");
   ObjectOutputStream objectOutputStream = new ObjectOutputStream(
     fileOutputStream);
   objectOutputStream.writeObject(man);

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  }

  /*
   * Deserializing Man's instance
   */
  Man manout = null;

  try {
   FileInputStream fileInputStream = new FileInputStream(
     "serialobject.ser");
   ObjectInputStream inputStream = new ObjectInputStream(
     fileInputStream);
   manout = (Man) inputStream.readObject();

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  } catch (ClassNotFoundException cnf) {
   // TODO Auto-generated catch block
   cnf.printStackTrace();
  }

 /*
  * Printing values from deserialized Man's object
  */
 System.out.println("Printing value of Deserailized Man's instance :");
 System.out.println("Gender: " + manout.getGender());
 System.out.println("Color: " + manout.getColor());
 System.out.println("Name: " + manout.getName());
 System.out.println("Address: " + manout.getAddress());

 }

}
```

Output: Here is expected output, printing values for deserialized object.



```
Printing value of Deserailized Man's instance :
Gender: Male
Color: Black
Name: Anderw
Address: Delhi
```

(2) If Super class is not Serializable but subclass is

In case super class is not Serializable than to serialize the subclass's object we must implement serializable interface in subclass explicitly. In this case the superclass must have a no-argument constructor in it. See the implementation below.

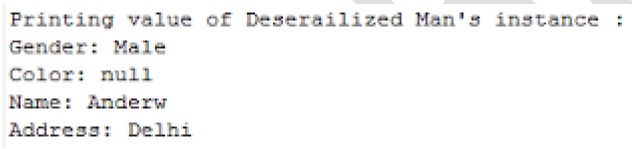
```java
package com.ashoksoft;

/**
 * @author Ashok Bollepalli
 */
public class Human {
 String gender;
 String color;

 Human() {
 }

 Human(String gender, String color) {
  this.gender = gender;
  this.color = color;
 }

 public String getGender() {
  return gender;
 }

 public void setGender(String gender) {
  this.gender = gender;
 }

 public String getColor() {
  return color;
 }

 public void setColor(String color) {
  this.color = color;
 }

}
```

```java
package com.ashoksoft;

import java.io.Serializable;

/**
 * @author Ashok Bollepalli
 */
public class Man extends Human implements Serializable {

 private static final long serialVersionUID = 1L;
 private String name;
 private String address;

 Man(String gender, String color, String name, String address) {
  super(gender, color);
  this.name = name;
  this.address = address;
 }

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public String getAddress() {
  return address;
 }

 public void setAddress(String address) {
  this.address = address;
 }

}
```

```java
package com.ashoksoft;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 * @author Ashok
 */
public class Implementation {
 public static void main(String args[]) {

  /*
   * Assigning values to Man class's instance
   */
  Man man = new Man("Male", "Black", "Anderw", "Delhi");

  /*
   * Serializing Man's instance
   */
  try {
   FileOutputStream fileOutputStream = new FileOutputStream(
     "serialObject.ser");
   ObjectOutputStream objectOutputStream = new ObjectOutputStream(
     fileOutputStream);
   objectOutputStream.writeObject(man);

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  }

  /*
   * Deserializing Man's instance
   */
  Man manout = null;

  try {
   FileInputStream fileInputStream = new FileInputStream(
     "serialobject.ser");
   ObjectInputStream inputStream = new ObjectInputStream(
     fileInputStream);
   manout = (Man) inputStream.readObject();

  } catch (FileNotFoundException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (IOException ioe) {
   // TODO Auto-generated catch block
   ioe.printStackTrace();
  } catch (ClassNotFoundException cnf) {
```

```
            // TODO Auto-generated catch block
            cnf.printStackTrace();
        }

        /*
         * Printing values from deserialized Man's object
         */
        System.out.println("Printing value of Deserailized Man's instance :");
        System.out.println("Gender: " + manout.getGender());
        System.out.println("Color: " + manout.getColor());
        System.out.println("Name: " + manout.getName());
        System.out.println("Address: " + manout.getAddress());

    }

}
```

Output: Here is expected output, printing values for deserialized object.



```
Printing value of Deserailized Man's instance :
Gender: Male
Color: null
Name: Anderw
Address: Delhi
```

If superclass is not Serializable then all values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process. As we can see in the output figure above 'Gender' gets a default value and 'Color' is getting null because of no default value set.

(3) If the superclass is serializable but we don't want the subclass to be serialized.

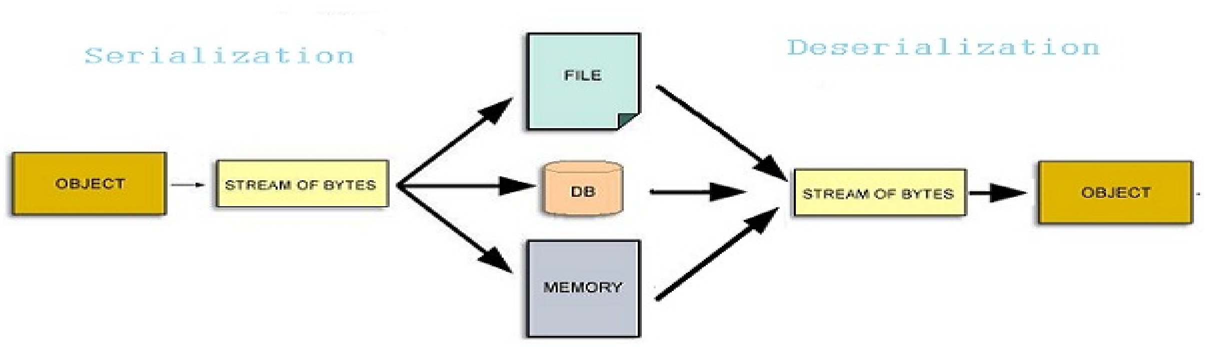
To prevent subclass from being serialized we must implement writeObject() and readObject() method and need to throw NotSerializableException from these methods.

Understanding the serialVersionUID

As per **java docs**, during serialization, runtime associates with each serializable class a version number, called a serialVersionUID, which is used during de-serialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then de-serialization will result in an InvalidClassException. A serializable class can declare its own serialVersionUID explicitly by declaring a field named "serialVersionUID" that must be static, final, and of type long:

private static final long serialVersionUID = 4L;

Here, the serialVersionUID represents your class version, and you should increment it if the current version of your class is modified such that it is no longer backwards compatible with its previous version.



Bullet Points

- Defining a ***serialVersionUID*** field in serializable class is **not mandatory**.
- If a serializable class has an explicit ***serialVersionUID*** then this field should be of type ***long*** *and must be static and final.*
- If there is no ***serialVersionUID*** field defined explicitly then serialization runtime will calculate default value for that class. The value can vary based on compiler implementation. Hence it is advisable to define ***serialVersionUID.***
- It is advised to use private access modifier for ***serialVersionUID.***
- Different class can have same ***serialVersionUID.***
- Array classes cannot declare an explicit serialVersionUID, so they always have the default computed value, but the requirement for matching serialVersionUID values is waived for array classes.
- If there is a difference between serialVersionUID of loaded reciever class and corresponding sender class then **InvalidClassException** will be thrown.
- You should use different ***serialVersionUID*** for different version of same class if you want to forbid serialization of new class with old version of same class.

JDBC

What is JDBC?

JDBC is a Java API that allows Java programs to access database management systems (Relational database). The JDBC API consists of a set of interfaces and classes which enables java programs to execute SQL statements.

Why we need JDBC?

All applications requires some persistent mechanism to store application generated data. In java we can store data using variables and Objects. But these variables and Objects is will be stored into Secondary memory. This secondary memory will be available till the application is executing. Once application execution completed this secondary memory will be vanished, then data which is stored in secondary memory also will get vanished. So we will lose the data if store in variables and Objects. To overcome this problem we need to go for Persistence stores.

What is Persistence and what is Persistence store?

The process of storing and maintaining the data for long time is called Persistence. To store and maintain data for long time we need Persistence stores. Below are the Persistence stores available

1. File 2. RDBMS

Files are used to store small amount of data.
Databases are used to store huge amount of data and having lot of advantages when compared with the files.

After storing the data into Persistence stores, we can perform some operations on Persistent data, these operations are called as Persistence operations. Below are the persistence operations

CURD or CRUD or SCUD

C- Create /Insert

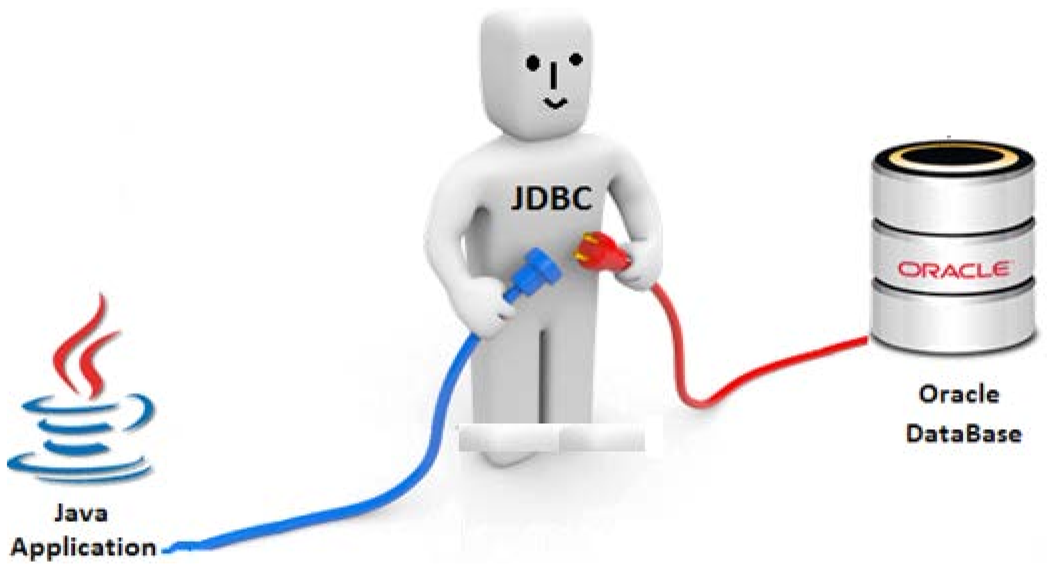
U – Update

R – Retrieve

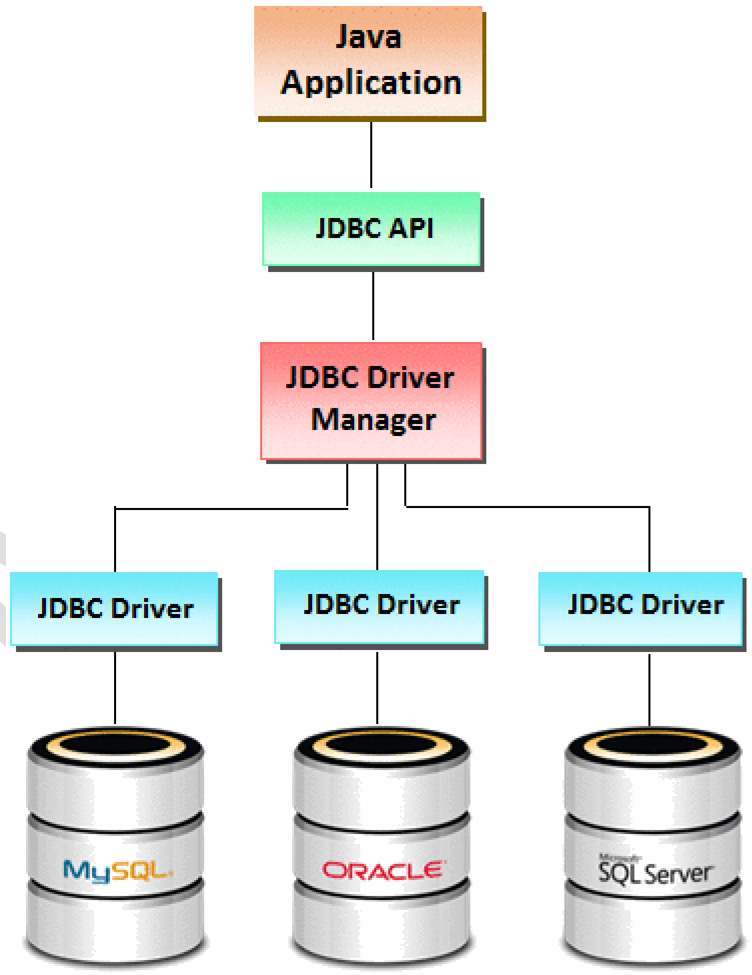
D – Delete

To perform these persistence operations on persistence data we need Persistence Technologies.
JDBC is one of the Persistence Technology. It is released by Sun micro system as part of JDK 1.1v.

Java programs cannot communicate with the Databases directly. JDBC acts as a mediator between Java and Database.



The following is an Architecture of JDBC Applications



From the above architecture we have to developthe java application which communications with DB server.

JDBC core components

The JDBC API consists of the following core components:

1. JDBC Driver
2. Connections
3. Statements
4. ResultSets

1. JDBC Driver:

JDBC driver is a collection of classes which implements interfaces defined in the JDBC API for opening database connections, interacting with database and closing database connections.

2. Connections:

Before performing any database operation via JDBC, we have to open a database connection. To open a database connection we can call getConnection () method of DriverManager class.

3. Statements:

The JDBC statements are used to execute the SQL or PL/SQL queries against the database. We need a statement for every single query.

4. ResultSets:

A query returns the data in the form of ResultSet. To read the query result data, ResultSet provides a cursor that points to the current row in the result set.

Steps to connect with database in java using JDBC

1. Load and Register the JDBC driver.

2. Get the connection by using Database details (url, username and password).

3. Create the statement.

4. Prepare and execute the Query.

5. Close database connection.

1. Load the JDBC driver:

First step is to load or register the JDBC driver for the database. Predefined class **Class** provides forName() method to dynamically load any class into JVM memory

What is Driver class?

A class which provides the implementation of JDBC Driver interface is called as Driver class.

The name of the driver class vary from Database to Database

Syntax : Class.forName("DriverClassName");

Ex : To load Oracle Driver Class we will use below statement

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2. Create connection:

Second step is to open a database connection. DriverManager class provides the facility to create a connection between a database and the appropriate driver.To open a database connection we can call getConnection() method of DriverManager class.

For getConnection() method we need to pass three parameters.

url, username and password

url: url is used to specify in which computer database server software is installed, and we need to specify port number, Service name also.

username and password: username and password are used for authentication purpose.

```
public static Connection getConnection(String url,String name,String password)
throws SQLException
```

Example to establish connection with the database

```
Connection con=new DriverManager.getConnection(url, username, password);
Example:
```

3. Create statement:

The statement object is used to send SQL queries to the database .To create a statement object we have to call createStatement() method of Connection interface.

Syntax :

```
public Statement createStatement()throws SQLException
```

4. Execute statement:

Statement interface provides the methods to execute a statement.

Call any one of the following three methods of Statement interface is used to execute queries to the database and to get the output.

- **executeUpdate():** Used for non-select operations.
- **executequery():** Used for select operation.

5. Close database connection:

After done with the database connection we have to close it. Use close() method of Connection interface to close database connection. The statement and ResultSet objects will be closed automatically when we close the connection object.

Syntax:

```
public void close()throws SQLException
```

DriverManager class

- The DriverManager class acts as an interface between user and drivers
- It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
- The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method **DriverManager.registerDriver().**

Methods

1) **public static void registerDriver(Driver driver):**
2) **public static void deregisterDriver(Driver driver):**
3) **public static Connection getConnection(String url):**
4) **public static Connection getConnection(String url,String uname,String pwd)**

Note : Whenever we open a database connection, its mandatory that we have to close the connection, because every database will have limited no.of connections.

Requirement #1:

Develop a java program to create a table in the database server.

```
package info.ashok;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

/**
 * @abstract This class is used to Create a table in DB

 *
 */
public class DBOperations {
        public static void main(String args[]) {
                Connection conn = null;
                Statement statement = null;
                String query = "create table EMPLOYEE("
                                + "EMPLOYEE_ID NUMBER(5) NOT NULL, "
                                + "NAME VARCHAR(20) NOT NULL, "
                                + "SALARY NUMBER(10) NOT NULL, "
                                + "PRIMARY KEY (EMPLOYEE_ID) )";
                String url = "jdbc:oracle:thin:@localhost:1521/XE";
                String uname = "ashok";
                String pwd = "ashok";
                try {
                        // Load the driver (Using Oracle Driver)
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        // get connection
                        conn = DriverManager.getConnection(url, uname, pwd);
                        // create statement
                        statement = conn.createStatement();
                        // execute query
                        statement.execute(query);
                        // close connection
                        statement.close();
                        conn.close();
                        System.out.println("Table created successfully.");
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Note : set the CLASSPATH for ojdbc14.jar file and Execute the program like below

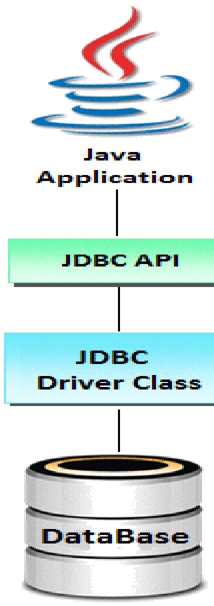
```
C:\> set CLASSPATH=ojdbc14.jar;.;

C:\> javac DBOperations.java

C:\> java DBOperatons
```

JDBC driver

Jdbc API contains a set of classes and Interfaces where classes definition is provided by Sun Micro System and Interfaces Implementation are not provided. For interface Implementation classes will be provided by vendors, these set of classes are called **Jdbc Driver Software**.

A Driver software contains set of classes among these classes one class is driver class.

Driver class is a mediator class between a java application and a database.



Java Application use Jdbc API and this API connect to driver class, then driver class connect to database.

Database queries are classified into two types they are

1. **Select queries** 2. **Non-select queries**

What are non-select queries?

The query's which does not start with select keyword are called as non-select queries

(insert, update, delete, create, and drop etc).

To execute the non-select queries use a method executeUpdate().

Syntax: int executeUpdate (String sqlQuery);

What are the select queries?

The query's which starts with select keyword is called as select query.

To execute the select queries use a method executeQuery().

Syntax : ResultSet executeQuery(String sqlQuery)

Requirement #2:

Develop a Java Program to Insert a record into EMPLOYEE table.

```
package info.ashok;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

/**
 * @abstract This class is used to Insert a record into Employee     table in DB
 * @author Ashok Bollepalli
 *
 */
public class DBOperations {
        public static void main(String args[]) {
                Connection conn = null;
                Statement statement = null;
                String query = "insert into EMPLOYEE "
                                + "(EMPLOYEE_ID, NAME, SALARY) "
                                + "values (1, 'Raju', 50000)";

                String url = "jdbc:oracle:thin:@localhost:1521/XE";
                String uname = "ashok";
                String pwd = "ashok";
                try {
                        // Load the driver (Using Oracle Driver)
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        // get connection
                        conn = DriverManager.getConnection(url, uname, pwd);
                        // create statement
                        statement = conn.createStatement();
                        // execute query
                        statement.executeUpdate(query);
                        // close connection
                        statement.close();
                        conn.close();
                        System.out.println("Record inserted successfully.");
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Requirement #3:

Develop a Java program to update a record in the DB table.

Execute the above program by replacing the query as given below

```
String query = "update EMPLOYEE set " + "NAME = 'Abhishek' "
                                + "where EMPLOYEE_ID = 1 ";
```

Requirement #4:

Develop a Java Program to delete a record in DB table.

Execute the above program (Requirement: #2) by replacing the query as given below;

```
String query = "delete EMPLOYEE " + "where EMPLOYEE_ID = 1 ";
```

Requirement #5:

Develop the java application to retrieve the records from EMPLOYEE table and display to the client.

- > To retrieve the records from table we have to use select queries.

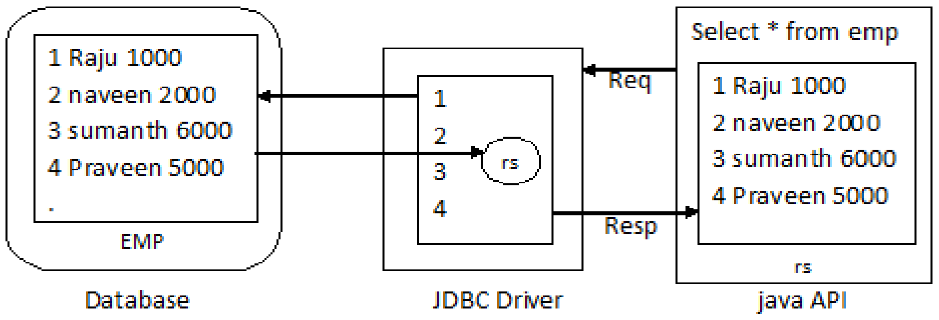
- > To execute the select queries we have to use executeQuery(String sql) method.

- > JDBC Driver will send the SQL query to database, in Database Database Engine will execute

The query and returns the results to JDBC Driver

- > After receiving the results from Database, it is the responsibility of JDBC Driver to convert the results into Object format. That Object is called as ResultSet object and this ResultSet object will return to java application.

Ex: ResultSet rs = stmt.executeQuery (“SELECT * FROM EMPLOYEE”);



Whenever we got the result set object(rs). It is associated with a ResultSet pointer.

Initially this ResultSet pointer points before first row.

- > To get the fist row data we need to move ResultSet pointer to the row position.

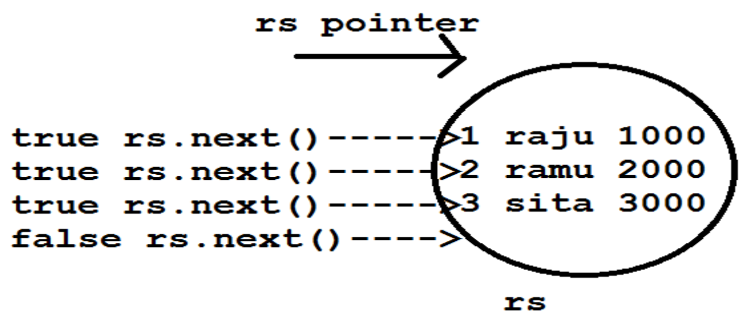
- > To move the ResultSet pointer we need to call a method next().

Syntax: boolean next()

When we call the next method

- > If the next record(row) is available it returns the true.

- > If the next record is not available it returns the false.


rs pointer
true rs.next() ----->1 raju 1000
true rs.next() ----->2 ramu 2000
true rs.next() ----->3 sita 3000
false rs.next() ---->
rs

If rs.next() method return type is true, that means record (row) is available.

Row : Collection of columns.

Now we have to get the columns data.

To get the data from columns ResultSet is providing some getterMethods(getXXX()) based on column data type.

If Column type is NUMBER associated getter method is getInt() method.

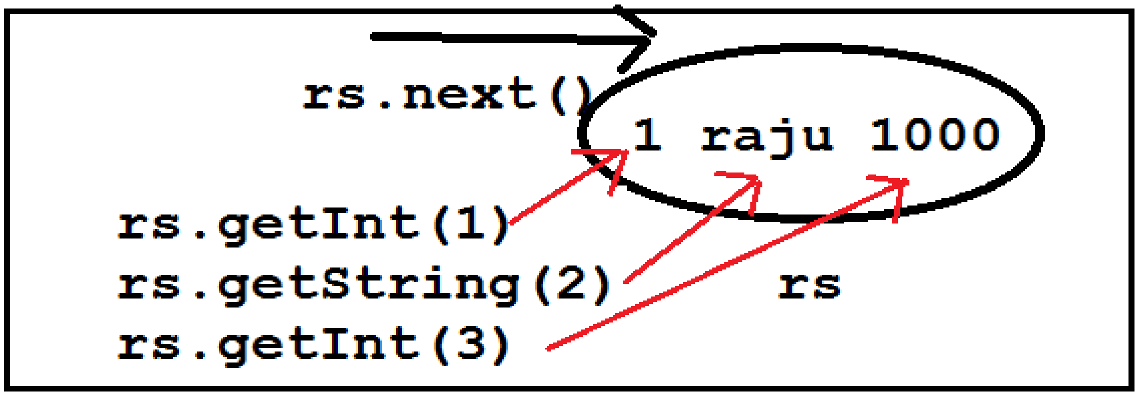
If Column type is VARCHAR associated getter method is getString() method.

Note : One Row may contains multiple columns, so retrieve particular column data we have to pass column index or column name as a parameter for getter methods.

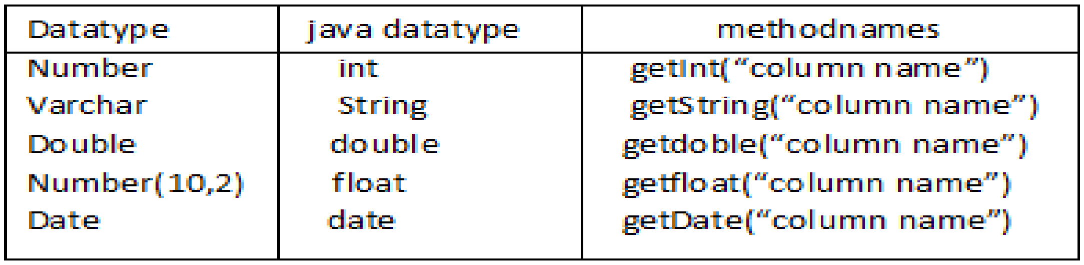
Syntax :

Int getInt(int columnIndex) or Int getInt(String columnName)

Note: Column index will start from 1.



The following diagram shows how to read the values of a specific data type.



Datatype	java datatype	methodnames
Number	int	getInt("column name")
Varchar	String	getString("column name")
Double	double	getdoble("column name")
Number(10,2)	float	getfloat("column name")
Date	date	getDate("column name")

JDBC PreparedStatement interface

The JDBC PreparedStatement is used to execute parameterized queries against the database. PreparedStatement is an interface which provides the methods to execute parameterized queries. A parameter is represented by ? symbol in JDBC. PreparedStatement extends the Statement interface. We can get a PreparedStatement object by invoking the prepareStatement() method of Connection interface.

Syntax: PreparedStatement pstmt = conn.prepareStatement(SQL);

Advantages of PreparedStatement

1. Parameterized query: Provides the facility of parameterized query.

2. Reusable: PreparedStatement can be easily used with new parameters.

3. Performance: It increases the performance because of database statement caching.

Difference between Statement and PreparedStatement in JDBC

Statement	PreparedStatement
1. Statement not executes the parameterized query.	1. PreparedStatement can execute the parameterized query.
2. Relational DB uses following 4 step to execute a query: a. Parse the query. b. Compile the query. c. Optimize/Plan the query. d. Execute the query. A statement always executes the all four steps.	2. Relational DB uses following 4 step to execute a query: a. Parse the query. b. Compile the query. c. Optimize/Plan the query. d. Execute the query. PreparedStatement pre-executes first three steps in the execution.
3. No database statement caching in case of statement.	3. It provides the database statement caching the execution plans of previously executed statements. Hence database engine can reuse the

	plans for statements that have been executed previously.

Note: PreparedStatement improves the performance of java application when compared with statement object.

By using PreparedStatement also we can perform CURD operations.

C-create a record, U-update a record, R-retrieve records, D-delete a record

- Register the JDBC Driver
- get the connection from database server.
- Create a PreparedStatement object by supplying query as input. These query must contain positional parameters.
- Ex: insert into emp values (?,?,?);
- Supply the values to positional parameters by using setxxx() methods.
- Execute the queries by calling executeUpadate() or executeQuery()
- Close the connection

Requirement #6:

Develop a java application to insert a record into EMPLOYEE table by using prepared statement

```
package info.ashok;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

/**
 * @abstract This class is used to Insert a record in Table


 */
public class DBOperations {
		public static void main(String args[]) {
				Connection conn = null;
				PreparedStatement pstmt = null;

				String url = "jdbc:oracle:thin:@localhost:1521/XE";
				String uname = "ashok";
				String pwd = "ashok";

				try {
						// get connection
						conn = DriverManager.getConnection(url, uname, pwd);
// Creating Query
String query = "insert into EMPLOYEE (EMPLOYEE_ID, NAME, SALARY) values (?,?,?)";
// create preparedStatement
pstmt = conn.prepareStatement(query);
        // set values
                pstmt.setInt(1, 1);
                pstmt.setString(2, "Bharat");
                pstmt.setInt(3, 62000);
                // execute query
                pstmt.executeUpdate();
                // close connection
                pstmt.close();
                conn.close();
                System.out.println("Record inserted successfully.");
        } catch (Exception e) {
                        e.printStackTrace();
        }
 }
}
```

Requriement #7:

Develop a java program to update a record in DB table using PreparedStatement.

Execute the above program by replacing with below query.

String query = "update EMPLOYEE set “+ "SALARY = ? " + "where EMPLOYEE_ID =? ";

Note: Set the values for positional parameters and execute the program.

Requriement #8:

Develop a java program to delete a record in DB table using PreparedStatement.

Execute the above program by replacing with below query.

String query = "delete from EMPLOYEE " + "where EMPLOYEE_ID = ?";

Note : Set the values for positional parameter and execute the program

Requirement #9:

Develop a java program to retrieve records from EMPLOYEE table.

```
package info.ashok;
```

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

/**
 * @abstract This class is used to Retrieve Records from TABLE.
 *
 *
 */
public class DBOperations {

        public static void main(String args[]) {
                Connection conn = null;
                PreparedStatement pstmt = null;

                String url = "jdbc:oracle:thin:@localhost:1521/XE";
                String uname = "ashok";
                String pwd = "ashok";

                try {

                        // loading the driver
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        // get connection
                        conn = DriverManager.getConnection(url, uname, pwd);

                // Creating Query
                String query = "select EMPLOYEE_ID, NAME from EMPLOYEE";
                        // create preparedStatement
                        pstmt = conn.prepareStatement(query);

                        //Execute the query
                        ResultSet rs = pstmt.executeQuery(query);

                        //Process the results
                        while (rs.next()) {
                                String empId = rs.getString("EMPLOYEE_ID");
                                String empName = rs.getString("NAME");
                                System.out.println("EmpId : " + empId);
                                System.out.println("EmpName : " + empName);
                        }
                        // close connection
                        pstmt.close();
                        conn.close();

                } catch (Exception e) {
                        e.printStackTrace();
                }
```

```
            }
    }
```

JDBC CallableStatement

The JDBC CallableStatement is used to execute the store procedure and functions. CallableStatement interface provides the methods to execute the store procedure and functions. We can get a statement object by invoking the prepareCall() method of Connection interface.

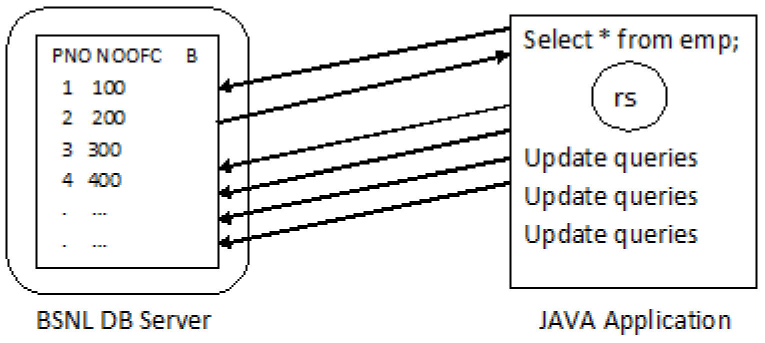
Syntax:

CallableStatement callableStatement = conn.prepareCall("{call procedurename(?,?...?)}");

Note: A store procedure is used to perform business logic and may return zero or more values. A function used to perform calculation and it can return only one value.

Use Case :

Calculate the bill amount to all the customers which are available in customer table.



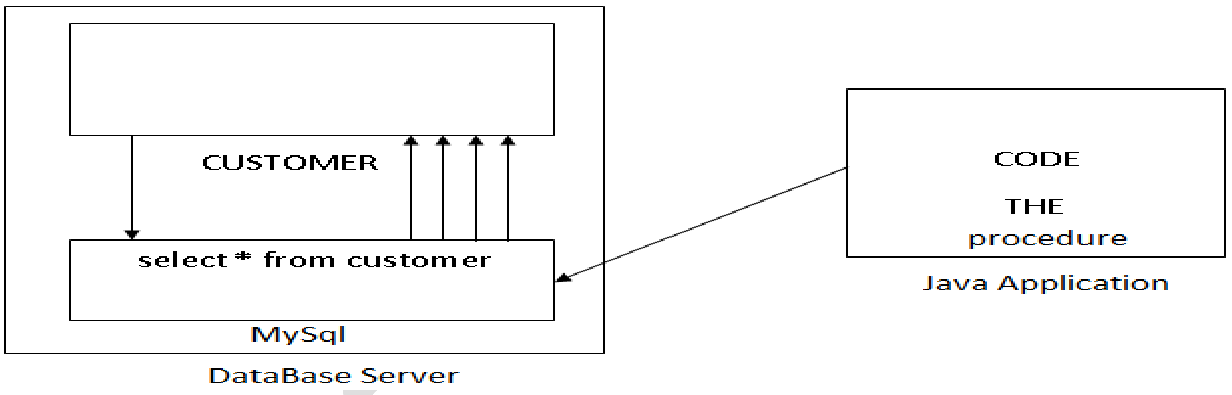
```
package info.ashok;

import java.sql.*;

public class DBOperations {
        public static void main(String[] args) throws SQLException {
                DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                Connection con = DriverManager.getConnection

                ("jdbc:oracle:thin:@localhost:1521:xe", "ashok", "ashok");
                String query = "Select * from BSNLcustomer";
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery(query);
                while (rs.next()) {
                        int noofc = rs.getInt("noofc");
                        double bamount = noofc * .10;
                        int phoneno = rs.getInt("phoneno");
```

```
                              String UQuery = "update BSNLcustomer set bamount = ? where phoneno =
?";
                              // System.out.println(UQuery);
                              PreparedStatement pstmt = con.prepareStatement(UQuery);
                              pstmt.setDouble(1, bamount);
                              pstmt.setInt(2, phoneno);
                              pstmt.executeUpdate();
                    }
          }
}
```

The disadvantage above approach we are interact with database server from multiple times through networks. Because of this we get the performance issue we can resolve this by using procedures.

We can write the procedures with business logic. Procedures in database server procedures run's inside database server.

Whenever anybody call processor, database server run's it procedures improves performance of project.



Procedures:

A stored procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again. So if you think about a query that you write over and over again, instead of having to write that query each time you would save it as a stored procedure and then just call the stored procedure to execute the SQL code that you saved as part of the stored procedure.

In addition to running the same SQL code over and over again you also have the ability to pass parameters to the stored procedure, so depending on what the need is the stored procedure can act accordingly based on the parameter values that were passed.

We can use three types of procedures

I. A procedure which doesn't take any type of parameters.
II. A procedure which takes input parameter.
III. A procedure input parameter as well as output parameter.

Create a procedure in DB and execute it

create or replace procedure MyProc

as begin

insert into emp values(2,'naveen',2000);

end MyProc;

Execution : exec MyProc;

Create a procedure which takes input parameter

Create or replace procedure MyProc(veno IN number, vname IN varchar2, vsalary in number)

as

begin

insert into emp values(veno, vname, vsalary);

end MyProc;

Execution :

To run the above procedure we use the following command exec MyProc(1,'abc',2345);

Create a procedure which take input and output parameters

create or replace procedure addition(no1 in number, no2 in number, result out number)

as begin

result := no1 + no2;

end addition;

/

Execution : exec addition(10,20, :result); print result;

To call the above procedure we have to perform the following steps

Step 1: Before we call the procedure we must register / create variable. (/ slash)

Step 2: Call the procedure by supplying bind variable as input.

Step 3: After procedure is executed we can get the value from out variable and display.

Requirement #10:

Develop a java application to call a procedure which doesn't take any parameter.

```
package info.ashok;
```

```

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;

/**
 * @abstract This class is used to call Procedure from DB
 * @author
 *
 */
public class DBOperations {

	public static void main(String args[]) {
		Connection conn = null;
		CallableStatement cstmt = null;

		String url = "jdbc:oracle:thin:@localhost:1521/XE";
		String uname = "ashok";
		String pwd = "ashok";
		try {
		// loading the driver
		Class.forName("oracle.jdbc.driver.OracleDriver");
		// get connection
		conn = DriverManager.getConnection(url, uname, pwd);

		// Calling procedure which doesnt take any parameters
			cstmt = conn.prepareCall("{call MyProc}");

		// close connection
		cstmt.close();
		conn.close();

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Requirement #11:

Develop a java application to call the procedure which takes Input parameters.

```
package info.ashok;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;

/**
```

```
 * @abstract This class is used to call Procedure from DB

 *
 */
public class DBOperations {

	public static void main(String args[]) {
		Connection conn = null;
		CallableStatement cstmt = null;

		String url = "jdbc:oracle:thin:@localhost:1521/XE";
		String uname = "ashok";
		String pwd = "ashok";
		try {
			// loading the driver
			Class.forName("oracle.jdbc.driver.OracleDriver");
			// get connection
			conn = DriverManager.getConnection(url, uname, pwd);

			// Calling procedure which takes IN parameters
			cstmt = conn.prepareCall("{call MyProc(?,?,?)}");
			cstmt.setInt(1, 1);
			cstmt.setString(2, "Ashok");
			cstmt.setInt(3, 20000);
			cstmt.execute();

			// close connection
			cstmt.close();
			conn.close();

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Requirement #12 :

Develop a java application to call a procedure which takes input parameter and output parameter

```
package info.ashok;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;

/**
 * @abstract This class is used to call Procedure from  DB
 *
 */
public class DBOperations {

	public static void main(String args[]) {
		Connection conn = null;
		CallableStatement cstmt = null;

		String url = "jdbc:oracle:thin:@localhost:1521/XE";
		String uname = "ashok";
		String pwd = "ashok";
		try {
			// loading the driver
			Class.forName("oracle.jdbc.driver.OracleDriver");
			// get connection
			conn = DriverManager.getConnection(url, uname, pwd);

// Calling procedure which takes IN parameters & gives OUT parameters
			cstmt = conn.prepareCall("{call addition(?,?,?)}");
			cstmt.setInt(1, 11);
			cstmt.setInt(2, 22);
			cstmt.registerOutParameter(3, Types.INTEGER);
			cstmt.execute();
			int result = cstmt.getInt(3);
			System.out.println(result);

			// close connection
			cstmt.close();
			conn.close();

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Note : When we register out parameter it is responsibility of JDBC driver to declare a variable. Once the procedure is executed the value will be stored in the variable

Functions

In oracle we can use functions. They are two types of function are available in oracle.

I. Predefined functions (or) aggregate functions
II. User defined functions

In oracle we have the following aggregate functions they are

i. count()
ii. min()

iii. max()
iv. sum()
v. avg() and etc.

To call function in oracle we use Queries for example

Select sum(sal) from emp;

Select max(sal) from emp; Select count(*) from emp;

Requirement #13 :

Develop a java application to find the number of records available in EMPLOYEE table

```
package info.ashok;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;


/**
 * @abstract This class is used to call function from DB
 * @author Ashok
 *
 */
public class DBOperations {

        public static void main(String args[]) {
                Connection conn = null;
                Statement stmt = null;

                String url = "jdbc:oracle:thin:@localhost:1521/XE";
                String uname = "ashok";
                String pwd = "ashok";
                try {
                // loading the driver
                Class.forName("oracle.jdbc.driver.OracleDriver");
                // get connection
                conn = DriverManager.getConnection(url, uname, pwd);

                // create statement
                stmt = conn.createStatement();

        // execute query (calling the function)
        ResultSet rs = stmt.executeQuery("select count(*) from EMPLOYEE");
                        if (rs.next()) {
                                System.out.println(rs.getInt("count"));
                        }
                        // close connection
                        stmt.close();
                        conn.close();

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Types of ResultSets:

They are two types of ResultSets in java. They are

I. Forward only ResultSet
II. Bi-Directional ResultSet

What is forwardonly ResultSet?

A ResultSet which can move only forward direction is called as Forwardonly ResultSet. By default in java we get Forwardonly ResultSet.

What is Bi-Directional ResultSet?

A ResultSet which can move Forward and Backward is called as Bi-Directional ResultSet.

Bi-Directional ResultSet can be achieved in both statement and prepared statement object. To get the Bi-Directional ResultSet we have to sulpply parameters to statement object (or) prepared statement object.

To create Bi-Directional ResultSet for statement object.

Syntax:

createStatement(ResultSet TYPE,ResultSet CONCURRENCY);

For ResultSet TYPE we can supply any of the following three values.

i. TYPE_FORWARD_ONLY ---→By Default
ii. TYPE_SCROLL_SENSITIVE
iii. TYPE_SCROLL_INSENSITIVE

For ResultSet Concurrency we can supply any of the following two values.

iv. CONCUR_READ_ONLY ---→By Default
v. CONCUR_UPDATABLE

What is Sensitive ResultSet?

When we develop a java application to retrieve records and get the result to java application and if somebody modify the data in database server, if it got reflected in java application we call it as Sensitive ResultSet.

What is In-Sensitive ResultSet?

After modify the data in database server if it is not reflecting in ResultSet object we call it as Insensitive.

When we use CONCUR_UPDATABLE when even we modify the data in ResultSet object it get reflected in database server.

Requirement 14:

Develop a java program to retrieve records from Table (using Bi-Directional Result Set)

```
package info.ashok;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

/**
 * @abstract This class is used to Create a table DB

 *
 */
public class DBOperations {

        public static void main(String args[]) {
                Connection conn = null;
                Statement stmt = null;

                String url = "jdbc:oracle:thin:@localhost:1521/XE";
                String uname = "ashok";
                String pwd = "ashok";
                try {
                        // loading the driver
                        Class.forName("oracle.jdbc.driver.OracleDriver");
                        // get connection
                        conn = DriverManager.getConnection(url, uname, pwd);

                        // create statement
                        stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                        ResultSet.CONCUR_READ_ONLY);

        // execut the Query
        ResultSet rs = stmt.executeQuery("select * from emp");
                        rs.next();
                        rs.next();
                        System.out.println(rs.getRow());
                        rs.previous();
                        System.out.println(rs.getRow());
                        rs.previous();
                        System.out.println(rs.getRow());
                        // close connection
                        stmt.close();
                        conn.close();

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

When we call the previous() method the ResultSet pointer is placed to old recored.

Absoulte(): This method is used to move the ResultSet pointer to a specific Record/Row. Ex: rs.absolute(5);

When the above line is executed to ResultSet pointer is placed at 5th Row.

We can use Bi-Directional ResultSet in case of prepared statement object also.

Ex: PreparedStatement pstmt = con.prepareStatement(“select * from emp”
,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

Requirement #15:

Develop a java program to retrieve records from Table (using Bi-Directional Result Set)

To develop a sensitive ResultSet program we must follow the follow three steps.

I. Make the ResultSet object as sensitive
II. Use the refreshRow() method
III. In the Query instead of * we must use column names.

```
package info.ashok;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

/**
 * @abstract This class is used to Create a table DB
 *
 *
 */
public class DBOperations {

        public static void main(String args[]) {
                Connection conn = null;
```

```
                    Statement stmt = null;

                    String url = "jdbc:oracle:thin:@localhost:1521/XE";
                    String uname = "ashok";
                    String pwd = "ashok";
                    try {
                    // loading the driver
                    Class.forName("oracle.jdbc.driver.OracleDriver");
                    // get connection
          conn = DriverManager.getConnection(url, uname, pwd);

 stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
 ResultSet rs = stmt.executeQuery("select eno, ename, sal from emp");
 while(rs.next()){
 //System.out.println("press any key to get the next result");
                    //System.in.read();
                    //System.in.read();
                    rs.refreshRow();
                              System.out.println(rs.getString(1));
                              System.out.println(rs.getString(2));
                              System.out.println(rs.getString(3));
                    }
                    // close connection
                    stmt.close();
                    conn.close();

                    } catch (Exception e) {
                              e.printStackTrace();
                    }
          }
 }
```

Whenever java program has encounter a refreshRow() it will check the data in database server and the data in ResultSet object same or not. If they are not same refreshRow() will update the data in ResultSet object.

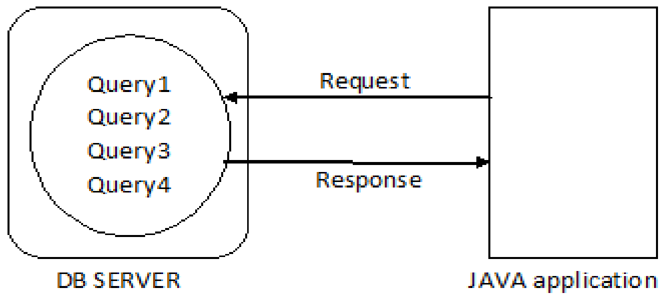
refreshRow() method work only from sensitive ResultSet.

In the SQL query instead of * we are specified column names. This is because whenever we change the specific column , the JDBC Driver update the specific column only.

Batch Updates

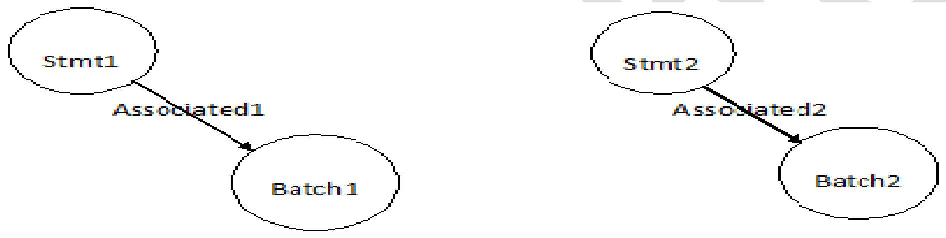
The JDBC batch processing provides the facility to execute a group of related queries. A group of related queries is known as a batch. It reduces the amount of communication overhead and hence improves the performance.

Instead of sending individual queries to Database, We can add multiple queries to Batch object and send Batch object to database server only once. So that we can reduce no.of iterations b/w Java program and DB Server. Because of these we can improve the performance.



When ever we create the statement object, immediately a Batch object will be created. This Batch object is associated with statement object.

Ex: Statement stmt1 = on.createStatement();
Statement stmt2 = con.createStatement();



Requirement : #16

Develop a java application to insert three records into EMPLOYEE table by using Batch updates

```
package info.ashok;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

/**
 * @abstract This class is used to Execute Batch Object

 *
 */
public class DBOperations {

        public static void main(String args[]) {
        Connection conn = null;
        Statement stmt = null;

        String url = "jdbc:oracle:thin:@localhost:1521/XE";
        String uname = "ashok";
        String pwd = "ashok";
        try {
        // loading the driver
                Class.forName("oracle.jdbc.driver.OracleDriver");
        // get connection
        conn = DriverManager.getConnection(url, uname, pwd);

        // create statement
        stmt = conn.createStatement();

        // add queries to Batch object
        stmt.addBatch("insert into emp values(122,'niranjan',16666)");
        stmt.addBatch("insert into emp values(124,'sravan',11143)");
        stmt.addBatch("insert into emp values(125,'mrudhu',14371)");

        // executing batch
        stmt.executeBatch();

        // close connection
        stmt.close();
        conn.close();

        } catch (Exception e) {
                        e.printStackTrace();
                }
        }
 }
```

By using object we can add insert, update and delete queries. By using Batch updates we can not perform select and create operations.

Syntax : int[] executeBatch();

Ex: int a[] = stmt.executeBatch();

The size of integer array is dependent on the size of the Batch object. The integer array contains the values which are got effected by each and every query. We can print these values by using a loop.

Ex:

```
for(int i = 0; i<a.length;i++){

        System.out.println(a[i]);

 }
```

While working with Batch updates if any query in the batch object is failed. JDBC driver throws java. Sql.BatchUpadeException.

Once if we sent the Batch object to database server we can clear the Batch by using cleared Batch method.

Transactions:

Transaction represents a single unit of work.

The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability.

I. **Atomicity** means either all successful or none.
II. **Consistency** ensures bringing the database from one consistent state to another consistent state.
III. **Isolation** ensures that transaction is isolated from other transaction.
IV. **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Every transaction is having two states.

1. Success state 2. Failure state

If all the steps in the transaction are executed successfully then we say transaction is success. If any one step is failed we can say transaction is failed.

When even we establish connection with database server it starts a transaction.

After establish the connection with database server. When the user perform any operation. The data will store permanently when we end the transaction.

When the JDBC driver starts the transaction. When ever we establish connection with database server immediately JDBC driver starts the transaction.

When we establish the connection internally JDBC driver uses "con.setAutoCommit(true); or

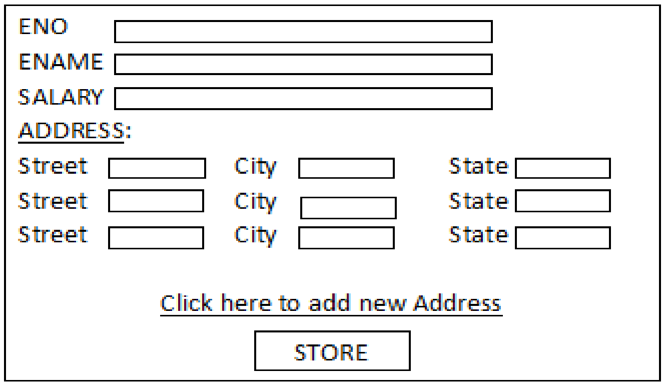
con.setAutoCommit(false);

and con.rollBack();".

The following an example of user defined transactions

```
 import java.sql.*;

public class AutoCommiteInsertRecords{

public static void main(String[] args)throws SQLException{

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

Connection con= DriverManager.getConnection("url","uname","pwd"); con.setAutoCommit(false);

Statement stmt = con.createStatement();

stmt.executeUpdate("insert into emp values(5,'Kiran',1111)");

stmt.executeUpdate("insert into emp values(6,'Naaga',2222)");

con.commit();
```

```
    }

    }
```



ENO					
ENAME					
SALARY					
ADDRESS:					
Street		City		State	
Street		City		State	
Street		City		State	

Click here to add new Address

STORE

The above application is used to capture employee details and store it in a DB. Our application should be able to support dealing with multiple address.

To achieve the above requirement we have two designs. Design1: In this design we create only table.

ENO	NAME	SALARY	STREET	CITY	STATE
1	Raju	10000	Ameetpet	HYD	AP
2	Raju	10000	Ameerpet	HYD	AP
---	-----	--------	-------------	-----	---
---	-----	--------	-------------	-----	---

The disadvantage of this approach is we find the duplicate data in the table. It is not recommend using design1.

Design2: In this design we will try to have two tables. They are emp and Address tables.

ENO	NAME	SALARY
1	Raju	10000

EMP table

ENO	STREET	CITY	STATE
1	Ameerp	HYD	AP
1	SRnagar	HYD	AP

ADDRESS table

MetaData:

Data about data is called as MetaData.

In JDBC we have three types of MetaData available. They are:

a. ResultSetMetaData
b. DatabaseMetaData
c. ParameterMetaData

ResultSetMetaData:

ResultSetMetaData is an object which gives more information about ResultSet object. By using ResultSetMetaData object we can find the number of column is available in ResultSet, the names of the columns, the Data types of the columns.

Ex: ResultSetMetaData rsmd = rs.getMetaData();

The following is example of using ResultSetMetaData object to find number of columns name of the columns and data types.

```
Ex:  rs = stmt.executeQuery("select * from emp");
ResultSetMetaData rsmd = rs.getMetaData();

System.out.println(rsmd.getColumnCount());

System.out.println(rsmd.getColumnName(2)); System.out.println(rsmd.getColumnTypeName(2));

System.out.println(rsmd.isSearchable(2));

System.out.println(rsmd.getColumnType(1));
```

DatabaseMetadata

DatabaseMetadata is an object which gives more information about underline Database server. That is it can find the database server, version information and JDBC Driver information.

```
DatabaseMetaData dbmd = con.getMetaData();
System.out.println(dbmd.getDatabaseMajorVersion());
System.out.println(dbmd.getDatabaseMinorVersion());
System.out.println(dbmd.getDatabaseProductName());
System.out.println(dbmd.getDriverMajorVersion());
System.out.println(dbmd.getDriverMinorVersion());
System.out.println(dbmd.getJDBCMajorVersion());
System.out.println(dbmd.getJDBCMinorVersion());
```

ParameterMetaData:

ParameterMetaData is an object which gives more information about ParameterMetaData's of PreparedStatement (possional parameter information is object). To get ResultSetMetaData object we take the help of ResultSet object. We use a method get MetaData to get ResultSetMetaData object.

```
ParameterMetaData pmd = pstmt.getParameterMetaData(); System.out.println(pmd.getParameterCount());
```

Working with BLOB and ClOB

BLOB and CLOB stand for Binary large object and Character large object respectively. Values of table columns declared using these types are stored independently of the table records. The size limit is determined by the external storage.

Suppose you want to allow the users to upload pictures to your web site. You will probably want to yous a BLOB column for storing those pictures.

Suppose you are writing a content management system and want to store product descriptions in your database. You don't want to impose max length limitations on the product description. So you will probably want to store these in a CLOB column.

Requirement #17:

Develop a java program to Store Image into Database.

Create a table in DB using below Query

SQL> create table empimg (eno number(5), name varchar2(20), image blob);

```
package info.ashok;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
 * @abstract This class is used to Insert image into table
 *
 *
 */
public class DBOperations {
		public static void main(String[] args) throws SQLException,FileNotFoundException {
				DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
				Connection con = 
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "ashok", "ashok");

				PreparedStatement pstmt = con.prepareStatement("insert into empimg 
values(?,?,?)");
				pstmt.setInt(1, 4);
				pstmt.setString(2, "anc");
				File f = new File("Jellyfish.jpg");
				FileInputStream fis = new FileInputStream(f);
				pstmt.setBinaryStream(3, fis, (int) f.length());
				pstmt.executeUpdate();
```

```
        }
}
```

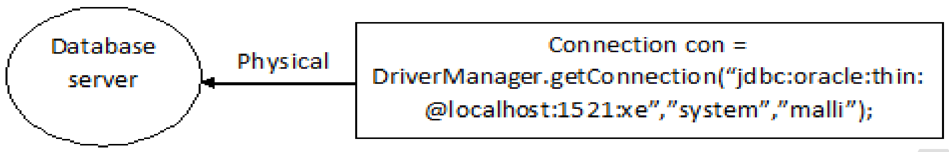
Requirement #18:

Develop a java program to retrieve Image from Database

```
package info.ashok;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * @abstract This class is used to retrieve Image from DB

 *
 */
public class DBOperations {
        public static void main(String[] args) throws 
SQLException,FileNotFoundException,IOException {
                DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                Connection con = 
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "ashok", "ashok");
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery("select * from empimg");
                rs.next();
                System.out.println(rs.getInt(1));
                System.out.println(rs.getString(2));
                byte imgarr[] =  rs.getBytes(3);
                FileOutputStream fos = new FileOutputStream("one.jpg"); // target folder to place 
img
                fos.write(imgarr);
                fos.close();
                con.close();
        }
}
```

Connection pool

When we develop a java application to get the connection from Database server. By using DriverManager.getConnection we always get physical connections.

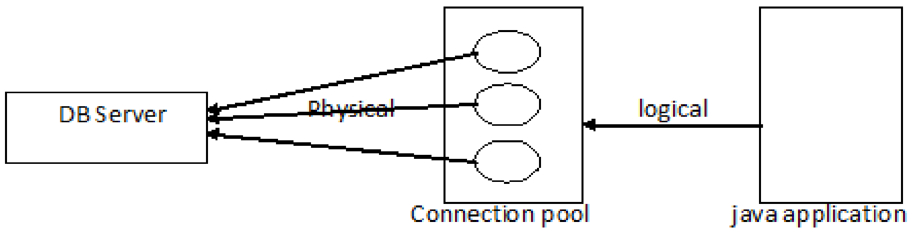

Database server
Physical
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","malli");

If we got a physical connection and we are not closing the connection after completion of work, the other application can not use the same connections.

Connection pool is java program which manages the connections of Database server. Connection pool contains set of connections.

There are so many connection pool programs are available some of them are.

i. DBCP (Database connection pool)
ii. C3p0 connection pool
iii. Weblogic connection pool
iv. JBOSS connection pool
v. Tomcat connection pool ……. Etc.



Generally connections pool program is released in the form of jar files. The jar file contains set of class.The meaning of using connection pool program is creating the object class and supply Driver class, url, username, password and initial capacity.

Requirement #19:

Develop a java program to establish Connection pool (Using DBCP connection pool)

```
package info.ashok;

import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;

import org.apache.commons.dbcp.BasicDataSource;

public class ConnectionPoolApp {
        public static void main(String args[]) throws IOException, SQLException {
                BasicDataSource bds = new BasicDataSource();
                bds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
                bds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
                bds.setUsername("ashok");
                bds.setPassword("ashok");
                bds.setInitialSize(3);

                Connection con1 = bds.getConnection();
                System.out.println(con1);
                System.in.read();
                System.in.read();
                Connection con2 = bds.getConnection();
                System.out.println(con2);
                System.in.read();
                System.in.read();
                Connection con3 = bds.getConnection();
                System.out.println(con3);
                System.in.read();
                System.in.read();
                Connection con4 = bds.getConnection();
                System.out.println(con4);
                System.in.read();
                System.in.read();
                Connection con5 = bds.getConnection();
                System.out.println(con5);
                System.in.read();
                System.in.read();
        }
}
```

Note: set the classpath for below jar files and execute the program.

1.tomcat-dbcp.jar, ojdbc14.jar

Requirement #20 :

Develop a java program to establish Connection pool (Using C3PO connection pool)

```
package info.ashok;

import java.beans.PropertyVetoException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class ConnectionPoolApp {
        public static void main(String args[]) throws IOException,
SQLException,ClassNotFoundException,PropertyVetoException {
                ComboPooledDataSource cpds = new ComboPooledDataSource();
                cpds.setDriverClass("oracle.jdbc.driver.OracleDriver");
                cpds.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
                cpds.setUser("ashok");
                cpds.setPassword("ashok");
                cpds.setInitialPoolSize(3);
                Connection con = cpds.getConnection();
                System.out.println("Connection:" +con);
                System.in.read();
                System.in.read();

        }
}
```

Note : Set the CLASSPATH for below jar fies and execute the program

2.13 RowSet Interfcae

ResultSet Disdavantages

- ResultSet Object is not serilaizable , because it is always maintains a connection with DB
- We can't pass theResultset object from one class to other class across the network.

Rowset

- **RowSet extends the ResultSet interface** so it has all the methods of ResultSet
- RowSet **can be serialized** because it doesn't have a connection to any database
- RowSet **Object can be sent from one class to another across the network**.

We have following types of RowSets

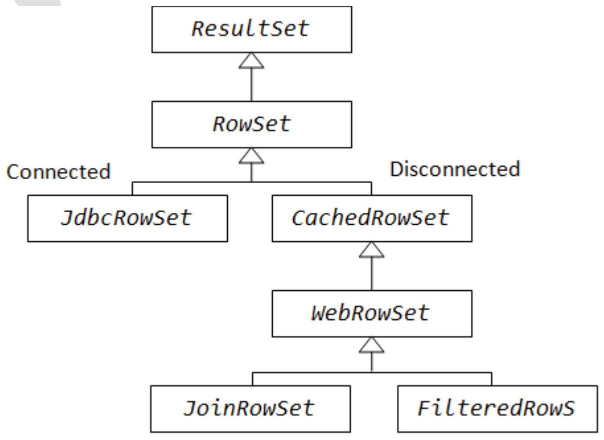
1. JdbcRowSet

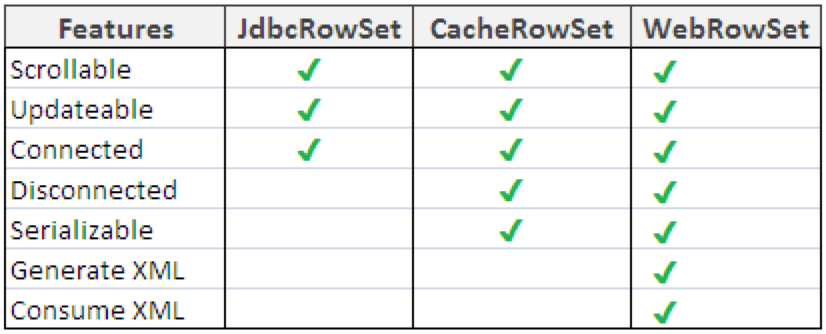
2. CachedRowSet

3. WebRowSet

4. JoinRowSet

5. FilteredRowSet





Features	JdbcRowSet	CacheRowSet	WebRowSet
Scrollable	✓	✓	✓
Updateable	✓	✓	✓
Connected	✓	✓	✓
Disconnected		✓	✓
Serializable		✓	✓
Generate XML			✓
Consume XML			✓

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
rowSet.setUsername("system");

rowSet.setPassword("oracle");
rowSet.setCommand("select * from emp400");
```

Requirement #21:

Develop a java program to read data from DB table and write it to Excel file

Many a time, a software application is required to generate reports in Microsoft Excel file format. Sometimes, an application is even expected to receive Excel files as input data. For example, an application developed for the Finance department of a company will be required to generate all their outputs in Excel.

Any Java programmer who wants to produce MS Office files as output must use a predefined and read-only API to do so.

What is Apache POI?

Apache POI is a popular API that allows programmers to create, modify, and display MS Office files using Java programs. It is an open source library developed and distributed by Apache Software Foundation to design or modify Microsoft Office files using Java program. It contains classes and methods to decode the user input data or a file into MS Office documents.

Components of Apache POI

Apache POI contains classes and methods to work on all OLE2 Compound documents of MS Office. The list of components of this API is given below.

i. POIFS (Poor Obfuscation Implementation File System) : This component is the basic factor of all other POI elements. It is used to read different files explicitly.
ii. HSSF (Horrible Spreadsheet Format) : It is used to read and write xls format of MS-Excel files.
iii. XSSF (XML Spreadsheet Format) : It is used for xlsx file format of MS-Excel.
iv. HPSF (Horrible Property Set Format) : It is used to extract property sets of the MS-Office files.
v. HWPF (Horrible Word Processor Format) : It is used to read and write doc extension files of MS-Word.
vi. XWPF (XML Word Processor Format) : It is used to read and write docx extension files of MS-Word.
vii. HSLF (Horrible Slide Layout Format) : It is used for read, create, and edit PowerPoint presentations.

viii. HDGF (Horrible DiaGram Format) : It contains classes and methods for MS-Visio binary files.
ix. HPBF (Horrible PuBlisher Format) : It is used to read and write MS-Publisher files.
x. This tutorial guides you through the process of working on Excel files using Java. Therefore the discussion is confined to HSSF and XSSF components.

Note: Older versions of POI support binary file formats such as doc, xls, ppt, etc. Version 3.5 onwards, POI supports OOXML file formats of MS-Office such as docx, xlsx, pptx, etc.

Like Apache POI, there are other libraries provided by various vendors for Excel file generation. These include Aspose cells for Java by Aspose, JXL by Commons Libraries, and JExcel by Team Dev.

Let us assume the following employee data table called emp_tbl is to be retrieved from the ORACLE database.

EMP ID	EMP NAME	DEG	SALARY	DEPT
1201	Gopal	Technical Manager	45000	IT
1202	Manisha	Proof reader	45000	Testing
1203	Masthanvali	Technical Writer	45000	IT
1204	Kiran	Hr Admin	40000	HR
1205	Kranthi	Op Admin	30000	HR

Use the following code to retrieve data from a database and insert the same into a spreadsheet.

```
import java.io.File;
import java.io.FileOutputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
public class ExcelDatabase
{
  public static void main(String[] args) throws Exception
  {
    Class.forName("Oracle.jdbc.driver.OracleDriver");
    Connection connect = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/XE" ,
"ashok" , "ashok");
    Statement statement = connect.createStatement();
    ResultSet resultSet = statement.executeQuery("select * from emp_tbl");
    XSSFWorkbook workbook = new XSSFWorkbook();
    XSSFSheet spreadsheet = workbook.createSheet("employe db");
    XSSFRow row=spreadsheet.createRow(1);
    XSSFCell cell;
    cell=row.createCell(1);
```

```
      cell.setCellValue("EMP ID");
      cell=row.createCell(2);
      cell.setCellValue("EMP NAME");
      cell=row.createCell(3);
      cell.setCellValue("DEG");
      cell=row.createCell(4);
      cell.setCellValue("SALARY");
      cell=row.createCell(5);
      cell.setCellValue("DEPT");
      int i=2;
      while(resultSet.next())
      {
        row=spreadsheet.createRow(i);
        cell=row.createCell(1);
        cell.setCellValue(resultSet.getInt("eid"));
        cell=row.createCell(2);
        cell.setCellValue(resultSet.getString("ename"));
        cell=row.createCell(3);
        cell.setCellValue(resultSet.getString("deg"));
        cell=row.createCell(4);
        cell.setCellValue(resultSet.getString("salary"));
        cell=row.createCell(5);
        cell.setCellValue(resultSet.getString("dept"));
        i++;
      }
      FileOutputStream out = new FileOutputStream(
      new File("exceldatabase.xlsx"));
      workbook.write(out);
      out.close();
      System.out.println("exceldatabase.xlsx written successfully");
    }
  }
```

Interview Questions

1. What is JDBC API and when do we use it?
2. What are different types of JDBC Drivers?
3. How can you load the drivers?
4. How does JDBC API helps us in achieving loose coupling between Java Program and JDBC Drivers API?
5. What is JDBC Connection? Explain steps to get Database connection in a simple java program.
6. What is the use of JDBC DriverManager class?
7. How to get the Database server details in java program?
8. What are the types of statements in JDBC?
9. What is JDBC Statement?

10. What is the difference between executeQuery, executeUpdate?
11. What is JDBC PreparedStatement?
12. What are the benefits of PreparedStatement over Statement?
13. What is JDBC ResultSet?
14. What are different types of ResultSet?
15. Differentiate between TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE.
16. How to use JDBC API to call Stored Procedures?
17. What is JDBC Batch Processing and what are it's benefits?
18. What does setAutoCommit do?
19. What is JDBC Transaction Management and why do we need it?
20. How to rollback a JDBC transaction?
21. What is JDBC Savepoint? How to use it?
22. What is Metadata and why should I use it?
23. What is JDBC DataSource and what are it's benefits?
24. How to achieve JDBC Connection Pooling using JDBC DataSource ?
25. What is Apache DBCP API?
26. What are common JDBC Exceptions?
27. What is CLOB and BLOB datatypes in JDBC?
28. What do you understand by DDL and DML statements?
29. What is difference between java.util.Date and java.sql.Date?
30. How to insert an image or raw data into database?
31. How to invoke Oracle Stored Procedure with Database Objects as IN/OUT?
32. When do we get java.sql.SQLException: No suitable driver found?
33. What are JDBC Best Practices?
34. What is connection pooling ?
35. What is the advantage of Properties file in jdbc ?

Servlets

The basic aim of servlets is to develop web applications. Before servlets came into picture there was a specification called CGI. Servlets specification developed by SUN and released to the industry. Lot of server vendors came forward for implementing these specifications of servlets. Servlets are nothing but set of rules given by SUN in the form of Interfaces. The J2EE server vendors have developed their own classes by implementing the interfaces developed by SUN.

The collection of classes developed by server vendors and Interfaces developed by SUN are known as Servlet-API.

Introduction to important concepts of Internet and Web technologies

Internet is a global system of interconnected computer networks that connect various Domain Name System (DNS) servers, web hosting servers, home and business users, etc. Internet is a network of networks.

Internet Protocols

Lot of data transfer happens in an internet between computers and networks, and there should be some rules to regulate these transfers. A communications protocol is a system of digital rules for data exchange within or between computers.

Some of the important protocols of internet are

TCP (Transmission Control Protocol)

IP (Internet Protocol)

UDP (User Datagram Protocol)

HTTP (Hypertext Transfer Protocol)

FTP (File Transfer Protocol) and

SMTP (Simple Mail Transfer Protocol).

The Internet protocol suite is a set of communications protocols used for the Internet and similar networks. It is commonly known as TCP/IP, because it's most important protocols are the Transmission Control Protocol (TCP) and the Internet Protocol (IP), and they were the first networking protocols defined in this standard.

Transmission Control Protocol (TCP) is a connection-oriented protocol that provides reliable, ordered, error-checked delivery of packets.

Internet Protocol (IP) has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers.

UDP is a connection-less protocol that does not provide reliability, order or error-checking. UDP messages are referred to as datagrams and a datagram is defined as a basic transfer unit associated with a packet-switched network in which the delivery, arrival time, and order of arrival are not guaranteed by the network.

HTTP is the protocol to exchange or transfer hypertext. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.

The World Wide Web (abbreviated as WWW or W3) or the web is a system of interlinked hypertext documents accessed via the Internet. With a web browser, one can view web pages and navigate between them via hyperlinks.

FTP is a standard network protocol used to transfer files from one host to another host over a TCP-based network.

SMTP is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks.

IP Address

Websites and pages over the internet have a unique worldwide address called the IP addresses (e.g. 192.168.1.6), but a domain name (e.g. javajee.com) is used to refer to a website as it is easy to remember.

There are two versions of IP specification in use now: IPv4 and IPv6. IPv4 is the most widely used version currently. An IPv4 address contains 4 parts and each part will be a number between 0 and 255, e.g. 192.168.1.6. An IPv6 address has 8 parts and each part must be a hexadecimal number between 0 and FFFF, e.g. FF3:9B3D:999:0:FF12:420:20:444D.

Domain Name System (DNS)

Domain Name System (DNS) provides a mapping between domain names like javajee.com and IP addresses like 192.168.1.6. When we type a domain name like javajee.com in the browser, it will use DNS to resolve or find the IP address and browser then uses the IP address to connect to the correct machine.

Web application

A web application is any application software that runs on a server and we can access it using a web browser client. For instance, consider this website; you access it using a client browser, but it is deployed on a server located elsewhere.

Popular server side technologies used to create web applications include JSP/Servlets, ASP, PHP etc.

Web server

A web server is software that helps to deliver web content (web pages) to the clients (e.g. web browser) through the Internet using HTTP protocol. Some of the commonly used web servers are Apache web server, Microsoft Internet Information Services (IIS), Nginx Nginx (pronounced "engine x") and GWS (Google Web Server). Other older web servers include Jigsaw web server from W3C, Oracle web server and Xitami web and FTP server developed by iMatix Corporation.

Java web containers like Apache Tomcat also can act as a web server, but is usually used along with another web server like Apache server.

Web browser

A web browser (e.g. Internet Explorer, Google chrome, Mozilla Firefox and Opera web browser) can read HTML documents and compose them into visible or audible web pages. JSP (Java Server Pages) based on Java, ASP (Active Server Pages) based on .net or the open source PHP (PHP: Hypertext Preprocessor) pages, all finally generate html pages with other client side technologies like JavaScript and CSS, and is sent to the web browser. Try viewing the source of any web page through the view source option and you will only html, JavaScript or CSS, but no JSP, ASP or PHP.

URI, URL and URN

A uniform resource identifier (URI) is a string of characters used to identify a name of a resource. Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols. Schemes specifying a concrete syntax and associated protocols define each URI.

URI is an abstract concept and the two concrete forms of URI are

Uniform resource locator (URL), which is most common and is frequently referred to informally as a web address. E.g. http://www.ososys.com/internet-and-web-technologies.

Uniform resource name (URN), which is rarely seen, and was designed to complement URLs by providing a mechanism for the identification of resources in particular namespaces. E.g. (from wikipedia): The ISBN 0-486-27557-4 cites unambiguously a specific edition of Shakespeare's play Romeo and Juliet. The URN for that edition would be urn: isbn:0-486-27557-4. To gain access to this object and read the book, its location is needed, for which a URL would have to be specified.

Client side technologies: HTML, JavaScript and CSS

HTML, JavaScript and CSS are three important client side web technologies which can be understood by a web browser and are independent of the server side technologies like JSP, ASP or PHP. As you have seen, JSP, ASP or PHP pages are finally converted to client side technologies such as HTML, Javascript and CSS, and sent to browser.

HTML (Hypertext Mark-up Language) is a markup language used to mark-up the different elements of a web page like headings, paragraphs, tables, images etc. Without markup, the contents will just appear as normal text without any headings, paragraphs, tables or images. A web browser (e.g. Internet Explorer, Google chrome, Mozilla Firefox and Opera web browser) can read HTML documents and compose them into visible or audible web pages.

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation semantics (the look and formatting) of a document written in a markup language like HTML. CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the layout, colors, and fonts. You can also define the look and feel using HTML alone, but that is not a good practice. One of the advantages of using CSS is that you can change the look and feel of a page just by changing the style sheet and don't have to change the web page itself.

JavaScript (JS) is a dynamic programming language understood by web browsers. We can write client-side (browser) scripts using JavaScript to create dynamic content on web pages like scrolling text, expanding menu etc., interact with the user, validate user inputs at client side, communicate asynchronously with a server, and even alter the document content that is displayed. JavaScript is not related to Java through the name is similar and there is some syntax similarity. JavaScript is not limited to web browsers, but can be used in many areas like mobile and desktop applications. There are many JavaScript libraries that provide some JavaScript functions out of the box and we can just use them without writing them again. Popular such libraries include JQuery, Angular JS, Dojo Toolkit, Ext JS, Microsoft's Ajax library etc.

Container

Container provides runtime environment for Java2ee (j2ee) applications. A web container is a predefined application provided by a server, its takes care of Servlet and JSP.

In console based java applications a class which contains main method acts as a container for other classes.

Operations of Container

- Life Cycle Management
- Communication Support
- Multithreaded support
- Security etc.

Life cycle management:

Servlet and JSP are dynamic resources of java based web application. The Servlet or JSP will run on a server and at server side. A container will take care about life and death of a Servlet or JSP. A container will instantiate, Initialize, Service and destroy of a Servlet or JSP. It means life cycle will be managed by a container.

Communication Support

If Servlet or JSP wants to communicate with server than its need some communication logic like socket programming. Designing communication logic is increase the burden on programmers, but container act as a mediator between a server and a Servlet or JSP and provides communication between them.

Multithreading

A container creates a thread for each request, maintains the thread and finally destroy it whenever its work is finished.

Security

A programmer is not required to write security code in a Servlet/JSP. A container will automatically provide security for a Servlet/JSP.

HTTP

HTTP is a synchronous request-response application network protocol. Client sends a request specifying one of the seven HTTP methods, the location of the resource to be invoked, a set of optional headers and an optional message body. The server sends back a response containing the version of HTTP we are using, a response code, a description of the response code, a set of optional headers, and an optional message body. HTTP is the primary protocol for most web applications on the Internet, regardless of whether they are written in Java, PHP or ASP.

HTTP Versions

Important versions that need to be considered are HTTP/1.0 and HTTP/1.1.

HTTP/1.0 defines the basics protocol with some methods such as GET, POST and HEAD, and did not support informational status codes.

HTTP/1.1 defines seven HTTP methods GET, POST, HEAD, OPTIONS, TRACE, PUT, DELETE, and also add many headers and facilities to the original HTTP/1.0.

Basic Features of HTTP

There are three basic features that make HTTP a simple but powerful protocol:

HTTP is connectionless: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.

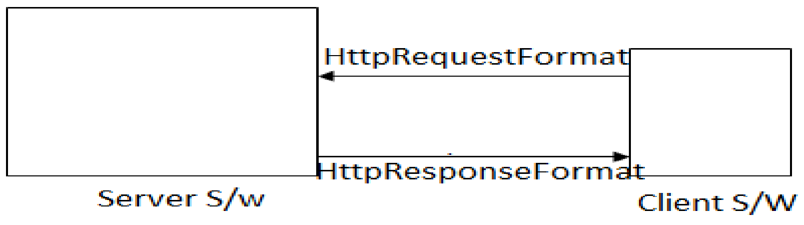
HTTP is media independent: It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.

HTTP is stateless: As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

HTTP protocol is divided into two parts. They are:

i. http Request format

ii. http Response format



The following are http Request format and Response format.

http Request format

Initial Request format
0 or more no.of headers
Blank line
Request body(optional)

http Response format

Initial Response format
0 or more no.of headers
Blank line
Response body(optional)

Initial Request Line : Initial Request Line is divided into following three parts. They are:

Method	Requested Resource	Protocol/Version

Method: The method indicates what operation has to be carried out by the server. The following are method of http protocol. They are:

GET, PUT, POST, DELETE, TRACE, HEAD, LOCATE // Server Methods

Request Resource: This is indicating the resource which has to be proceeding by the resource may be available or may not available.

Protocol/Version: Indicates which version of HTTP protocol is used by client.

GET	/four.html	HTTP/1.0

GET	/fout.html	HTTP/1.1

Header: The Header is used to send extra information about client to server. The following is header format.

Header-Name: Value (Send to extra information) (0 or more)

The following are the more important of two headers.

User-Agent

Accept-Language

User-Agent: Indicates which browser program has sent to request to server. To represent a next line character we use CR LF.

We can represent CR and LF by using \r\n characters.

Accept-Language: Header is use to specify what language is acceptable by the client based on the Accept-Language server will send the output to the client.

Initial Response Line:Initial Response line is divided into three parts.

Protocol/Version	Status Code	Status Message

Protocol/Version: indicates the protocol used by the server.

Status Code: indicates the status code of the Request send by the server.

HTTP protocol contains couple of status code.

100 to 199 Information

200 to 299 Successes

300 to 399 Re-Direct

400 to 499 Request Resources not available 500 to 599 Failed to execute Request Resource

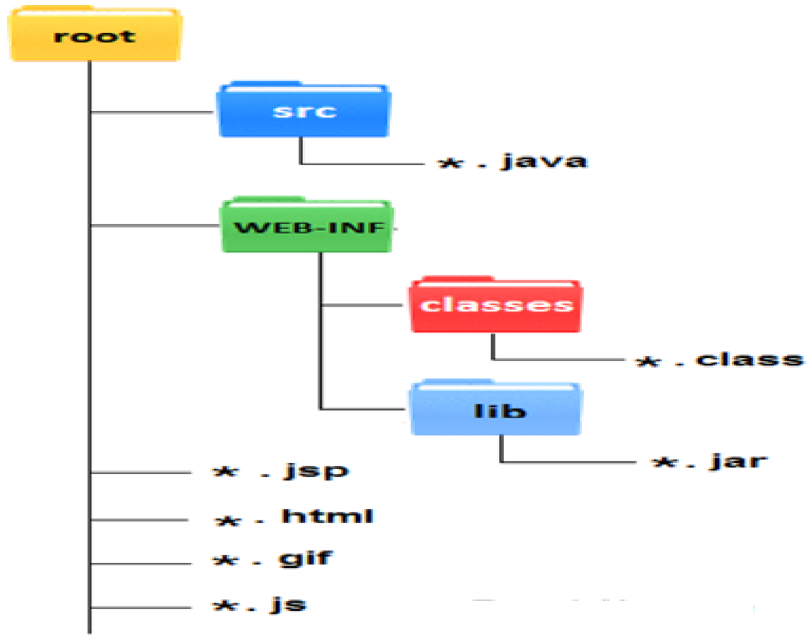
By using java we can develop both static and dynamic web-based applications.

Directory Structure of Web Application

For creating web application we should follow standard directory structure provided by sun Microsystem. Sun Microsystem has given directory structure to make a web application server independent.

According to directory structure, an application contain root folder with any name.

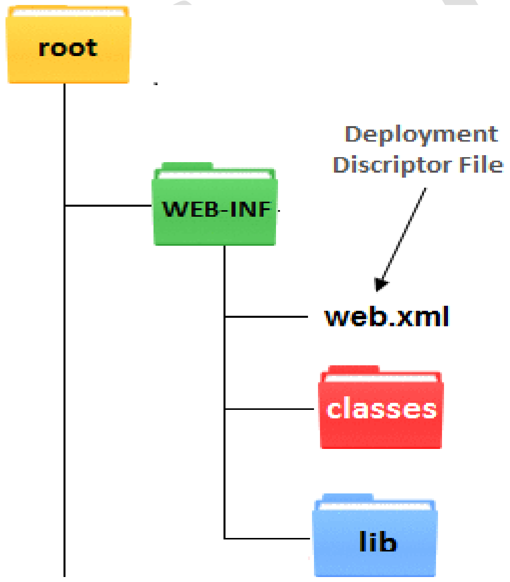
- Under root folder a sub folder is required with a name WEB-INF.
- Under WEB-INF two sub folder are required classes and lib.
- All jar files placed inside lib folder.
- Under root folder src folder are place for .java files
- Under root folder or under WEB-INF any other folders can exits.
- All image, html, .js, jsp, etc files are placed inside root folder
- All .class files placed inside classes folder.



Web.xml (Deployment Descriptor)

It is a web application deployment descriptor file, contains detail description about web application like configuration of Servlet, Session management, Startup parameters, Welcome file.etc. We cannot change the directory or extension name of this web.xml because it is standard name to recognized by container at run-time.

web.xml is present inside the Web-INF folder.



Servlet Components

The following are the most important “packages” to develop servlets. They are:

i. Javax.servlet
ii. Javax.servlet.http

The following are the most important interfaces and classes of javax.servlet package.

javax.servlet	
Interfaces	Classes
Servlet ServletRequest ServletResponse ServletOutputStream ServletConfig ServletContext RequestDispatcher	GenericServlet

The following are the most important interfaces and classes of javax.servlet.http package.

javax.servlet.http	
Interfaces	Classes
HttpServletRequest HttpServletResponse HttpSession	Cookie HttpServlet

Servlet

The servlet is an API, Released by Sun Micro System. The servlet is a java program which runs inside the Server.

The servlet is a java program which provides the ‘implementation’ of servlet interface directly or indirectly.

A servlet is a server side platform independent, dynamic and multithread java program, which runs in the context of server for extending the functionality of server. When multiple users make a request to the same servlet then all requests will be processed by the container by creating multiple threads for the same servlet.

Advantages of Servlet

- Better performance:
- Portability
- Robust

- Secure

Better performance: Because it creates a thread for each request not process (like CGI).

Portability: Because it uses java language and java is robust language.

Robust: Servlet are managed by JVM so no need to worry about memory leak, garbage collection etc.

Secure: Because it uses java language and java is a secure language.

Servlet as technology

As a technology servlet provides a model of communication between a web user request and the application or program on the web server.

Servlet as component

As a component servlet is a program which is executed in web server and responsible for dynamic content generation.

Main tasks of servlet

i. Read the implicit and explicit data sent by web browser.
ii. Generate result by processing the data.
iii. Send the implicit and explicit data as a response to the web browser.

Methods of servlet interface

1. **init(ServletConfig config):** It is used to initialize the servlet. This method is called only once by the web container when it loads the servlet.

Syntax: public void init(ServletConfig config)throws ServletException

2. **service(ServletRequest request,ServletResponse response):** It is used to respond to a request. It is called for every new request by web container.

Syntax: public void service(ServletRequest req,ServletResponse res)throws ServletException, IOException

3. **destroy():** It is used to destroy the servlet. This method is called only once by the web container when all threads of the servlet have exited or in a timeout case.

Syntax: public void destroy()

4. **getServletConfig():** It returns a servlet config object. This config object is passed in init method. Servlet config object contains initialization parameters and startup configuration for this servlet.

Syntax: public ServletConfig getServletConfig()

5. **getServletInfo():** It returns a string of information about servlet's author, version, and copyright etc.

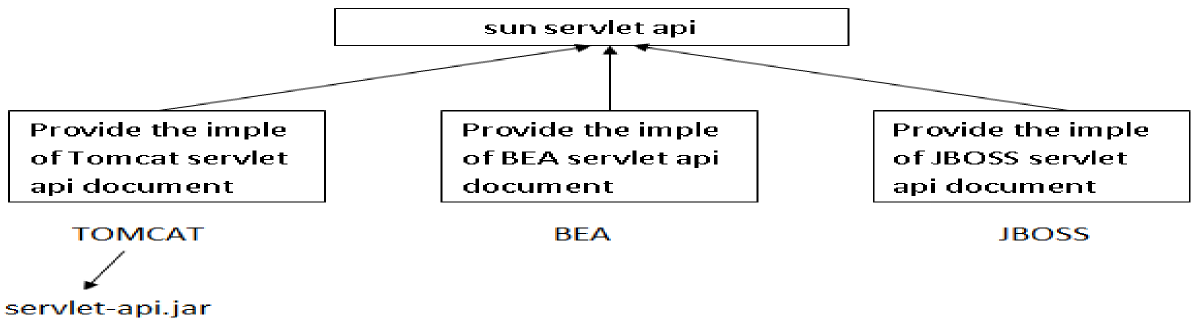
Syntax: public String getServletInfo()

Step 1 : Develop a class which provides the implementation of Servlet Interface.

```
package info.ashok;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class FirstServlet implements Servlet {
	ServletConfig config;

	public void init(ServletConfig config) {
		this.config = config;
		System.out.println("we are in init method");
	}

	public void service(ServletRequest request, ServletResponse response) {
		System.out.println("we are in service method");
	}

	public void destroy() {
		System.out.println("we are in destroy method");
	}

	public ServletConfig getServletConfig() {
		return config;
	}

	public String getServletInfo() {
		return "This is my First Servlet";
	}
}
```

Step 2: Compile the program

Note: To compile the program we need to set the CLASSPATH for the jar file which provided implementation for Servlet API


sun servlet api
Provide the imple of Tomcat servlet api document
Provide the imple of BEA servlet api document
Provide the imple of JBOSS servlet api document
TOMCAT
BEA
JBOSS
servlet-api.jar

jar file names vary between company to company in case of tomcat server the name of the jar file which provides to implementation of “**servlet-api.jar**”.

Step 3: Copy all servlet related .class files into classes folder(Which is in WEB-INF folder).
Step 4: Configure the servlet into web.xml file.

To configure a servlet file we need the following two xml tag.

- <Servlet>
- <Serlet-mapping>

<servlet>: tag are used for configure the servlet, Within this tag we write Servlet name and class name.

<Serlet-mapping>: tag are used for map the servlet to a URL .

The structure of web.xml files is already defined by sun MicroSystem. So as a developer we should follows the structure to develop the configuration of web resources.

While configuring a servlet in web.xml, three names are added for it.

- Alias name or register name
- Fully qualified class name
- url-pattern

```
<web-app>
<servlet>
<servlet-name>alias name</servlet-name>
<servlet-class>fully qualified class name</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>alias name</servlet-name>
<url-pattern>/url pattern</url-pattern>
</servlet-mapping>
</web-app>
```

Step 5: Deploy the project in any server.
Step 6: The following url which is used to access the first servlet.

http://localhost:8000/webapp/fs

It's mandatory that every servlet must be configure in "web.xml" file. When we deploy the project server reads the contents from 'web.xml' file.

Step 8: when we deploy any web-based application server will search for web.xml file. If the server found the xml file it will read the contents from web.xml file and store it in JVM's memory.

When we observe the above behavior it tomcat server. If it doesn't found web.xml it's not displaying any error message. Where as the web logic server has displayed an error message invalid application type.

To read the contents from xml files are required "parser" programs. In java they are two parser are available.

1. SAX parser
2. DOM parser

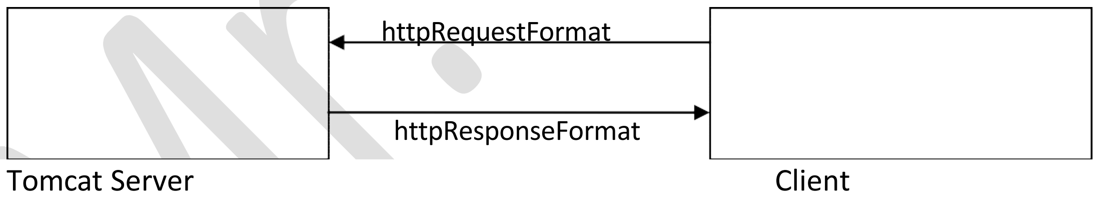
When deploy a project if the xml file is valid xml file server reads the contents and store it in JVM's memory. If the xml file is invalid SAX parser program throws an error message.

When we develop a project, server will not create any servlet objects.

How web.xml works

When a request comes it is matched with url pattern in servlet mapping attribute. In the above example all urls mapped with the servlet. You can specify a url pattern according to your need. When url matched with url pattern web server try to find the servlet name in servlet attributes same as in servlet mapping attribute. When match found control is goes to the associated servlet class.

When the client sends the request to server, server creates two objects. They are request object and response object. Now the server opens http request format and store the data into request object.



After the client has sent the request to server, server opens request and get the requested resource. Now the server checks is there any url pattern configured in web.xml file for that requested resource.

If it is available if does the remaining work. If it is not available in web.xml file server sends httpResponseFormat by using a status code you.

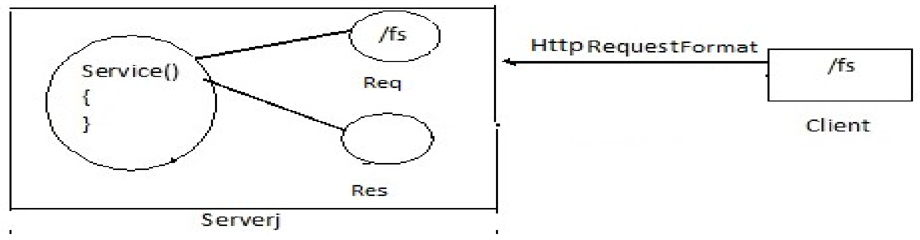
If the resource is available in web.xml file, server gets the servlet class name. Now the server checks is there any servlet object is available in the server. If not available it is the responsibility of server to create servlet object.

The server will try to load the class from class's folder. If the class file is not available in the classes folder, server throws an Exception java.lang.classnotFoundException.

If the class is loaded successfully server creates first servlet object and call the init().

Note: init() will be called only once when the servlet object is created.

Now the server will status the Exception service() to Execute the service() server supply request and response objects as parameters.



After the server has executed service() if it is the responsibility of server to send the response back to client. When the server sends the response to client, server removes request and response objects. The FirstServlet object remains same in the server.

It is the responsibility of server to remove servlet object. In the following two scenarios server will remove servlet objects.

When we stop the server.

When we undeploy the project.

Init(), service(), destroy() are called as servlet life cycle methods. In servlets the business logic will be written as part of service(), any code which has to be Executed only one time when the server create servlet object is provide in the init() method.

Servlet life cycle steps

i. **Load Servlet Class.**
ii. **Create Servlet instance.**
iii. **Call init() method.**
iv. **Call service() method.**
v. **Call destoy() method.**

1. Load Servlet Class: Web container loads the servlet when the first request is received. This step is executed only once at the time of first request.

2. Create Servlet instance: After loading the servlet class web container creates the servlet instance. Only one instance is created for a servlet and all concurrent requests are executed on the same servlet instance.

3. Call init() method: After creating the servlet instance web container calls the servlet's init method. This method is used to initialize the servlet before processing first request. It is called only once by the web container.

4. Call service() method: After initialization process web container calls service method. Service method is called for every request. For every request servlet creates a separate thread.

5.Call destoy() method: This method is called by web container before removing the servlet instance. Destroy method asks servlet to releases all the resources associated with it. It is called only once by the web container when all threads of the servlet have exited or in a timeout case.

If you observe the Servlet implementation clearly, we are writing our business logic only in service() method. But unnecessarily every time we need to implement remaining four methods also. It is time waste process and consumes time as well.

To simplify the development of servlets sun micro systems has given predefined classes. They are GenericServlet, HttpServlet. The following diagram shows the relationship between these classes.

Content Type

Content Type is also known as **MIME Type**. MIME stand for **Multipurpose internet Mail Extension**. It is a HTTP header that provides the description about what are you sending to the browser (like send image, text, video etc.).

This is the format of http protocol to carry the response contains to the client..

Example: Suppose you send html text based file as a response to the client the MIME type specification is

```
response.setcontentType("text/html");
```

MIME type have two parts, They are:

- Base name
- Extension name

Base name: It is the generic name of file.

Extension name: It is extension name for specific file type.

File	MIME Type	Extension
xml	text/xml.	.xml.
HTML	text/html.	.html.
Plaintext File	text/plain.	.txt.
PDF	application/pdf.	.pdf.
gif Image	image/gif.	.gif.
JPEG Image	image/jpeg.	.jpeg.
PNG Image	image/x-png.	.png.
MP3 Music File	audio/mpeg.	.mp3.
MS Word Document	application/msword.	.doc.
Excel work sheet	application/vnd.ms-sheet.	.xls.
Power Point Document	application/vnd.ms-powerpoint.	.ppt.

Can we have servlets with parameterized constructor?

We can have servlets with parameterized constructor, provided we also have a no argument one. This is because containers will instantiate servlets only using the no argument constructor. If we don't provide a constructor, java will provide a default contructor. However if we provide a constructor with arguments, container won't provide the default no arg one and hence won't be able to find a no-arg constructor and instantiate a servlets; and we will get a **java.lang.InstantiationException**.

GenericServlet

GenericServlet class Implements Servlet, ServletConfig and Serializable Interfaces. It provides the implementation of all the methods of these Interfaces except the service method. GenericServlet class can handle any type of request so it is protocol Independent. You may create a generic Servlet by inheriting the GenericServlet class and providing the Implementation of the service method.

Methods of GenericServlet

All methods of Servlet Interface are inherit in GenericServlet class because it Implements Servlet Interface. Following methods are used in GenericServlet.

Example of Servlet by inheriting the GenericServlet class

```
import java.io.*;
import javax.servlet.*;

public class GenericServletDemo extends GenericServlet
{
public void service(ServletRequest req,ServletResponse resp)
throws IOException,ServletException
{
res.setContentType("text/html");
PrintWriter out=resp.getWriter();
out.print("<html><body>");
out.print("<b>Example of GenericServlet</b>");
out.print("</body></html>");
}
}
```

HttpServlet

This is used to define httpServlet, to receive http protocal request and to send http protocol response to the client.

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace, doDelete, doOption etc.

1. doGet
2. doPost
3. doHead
4. doTrace
5. doDelete
6. doOption
7. doPut

```
public class HelloWorld extends HttpServlet {
        private static final long serialVersionUID = 1L;

        @Override
        public void service(ServletRequest request, ServletResponse response)
                        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello World example using HttpServlet class.</h1>");
                out.close();
        }
}
```

The above is the servlet which send html contents to the client. To send the html context to the client we have to write html tags as part of write() method or print() method.

The above servlet is trying to send html content to the client. Every time when the client send the request it is displaying same output. These type of applications are called as static web based applications.

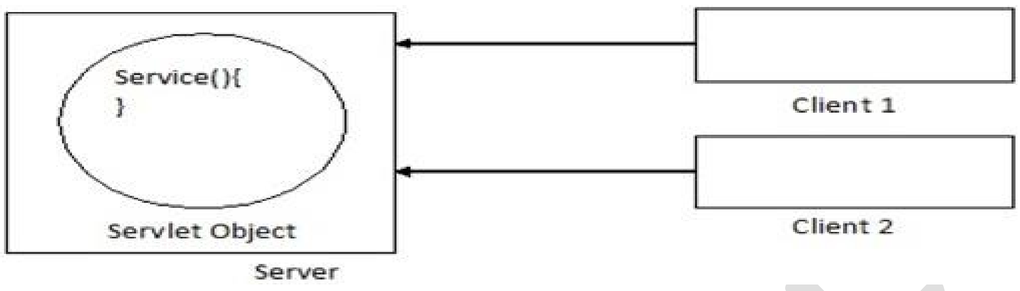
By using servlet also we can develop static applications as well as Dynamic applications. By using servlets it is not recommended to develop static web based applications. It's always recommended to develop Dynamic web based applications.

Instead of writing printwriter object we can use "Servlet.OutputStream". It is recommended to use ServletOutputStream to send binary data.

<u>Ex</u>:

```
ServletOutputStream out = response.getOutputStream();
out.Println("Welcome");
```

When multiple clients try to access one servlet only one servlet object will be created. This servlet object is shared between multiple clients.



The following is the servlet code which retrieve the data from data base server and display to the client.

```
public class RetrieveRecordsServlet Extends HttpServlet{
public void service( ------- ){
try{
// standard JDBC code
     ResultSet rs = stmt.ExecuteQuery("select * from product");
     PrintWriter out = response.getWriter();
}
}
```

When we deploy a project and if the server want to create the objects or server want to use the classes it will check in the following order.
Server checks for class files in classes folder.
If it is not available in classes folder server checks for project lib folder.

If it is not available in project lib folder, server check in server lib folder. If it is not available in these entire places server throws an Exception NoClassDefFound.

To secure the hard coding we are try to use command line arguments and system properties. But these will not work in servlets. This is because we can't supply command line arguments or system properties to the server. As the server is running the servlet application we can't use command line arguments or system properties. ServletContext, ServletConfig objects are used to remove hard coding. These objects are managed by servers(creation and removing the objects).

doPost() and doGet()

Http protocol mostly use either get or post methods to transfer the request. post method are generally used whenever you want to transfer secure data like password, bank account etc.

Get method	Post method

1	Get Request sends the request parameter as query string appended at the end of the request.	Post request send the request parameters as part of the http request body.
2	Get method is visible to every one (It will be displayed in the address bar of browser).	Post method variables are not displayed in the URL.
3	Restriction on form data, only ASCII characters allowed.	No Restriction on form data, Binary data is also allowed.
4	Get methods have maximum size is 2000 character.	Post methods have maximum size is 8 mb.
5	Restriction on form length, So URL length is restricted	No restriction on form data.
6	Remain in browser history.	Never remain the browser history.

<load-on-startup> Element

This is a **web.xml** configuration elements used to configure a servlet to create servlet object during startup time of the server or application deploy on server. It is also known as **pre initialization** of servlet. This element need integer value to configure.

Advantage of load-on-startup element

As we know well, servlet is loaded at first request. That means it consumes more time at first request. If we specify the load-on-startup in web.xml, servlet will be loaded at project deployment time or server start. So, it will take less time for responding to first request.

Passing positive value

If we pass the positive value, the lower integer value servlet will be loaded before the higher integer value servlet. In other words, container loads the servlets in ascending integer value. The 0 value will be loaded first then 1, 2, 3 and so on.

Let's try to understand it by the example given below:

web.xml

```
<web-app>

 <servlet>
  <servlet-name>servlet1</servlet-name>
  <servlet-class>com.tutorial4us.FirstServlet</servlet-class>
  <load-on-startup>0</load-on-startup>
 </servlet>
```

```
<servlet>
 <servlet-name>servlet2</servlet-name>
 <servlet-class>com.tutorial4us.SecondServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>
</web-app>
```

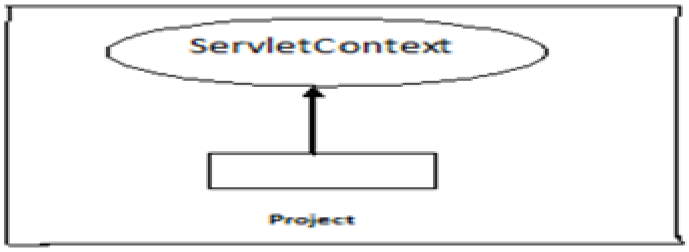
There are defined 2 servlets, both servlets will be loaded at the time of project deployment or server start. But, servlet1 will be loaded first then servlet2.

Passing negative value

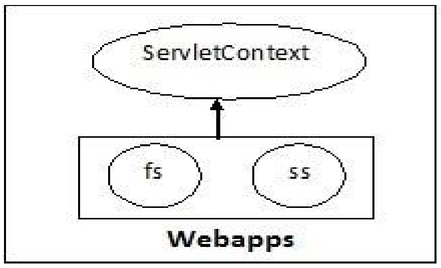
If a negative value is configure then a container ignores this tag and waits for first request to create an object of a servlet but a container does not throws any exception.

ServletContext

ServletContext is one of pre-defined interface available in javax.servlet.*; Object of ServletContext interface is available one per web application. An object of ServletContext is automatically created by the container when the web application is deployed.
Assume there exist a web application with 2 servlet classes, and they need to get some technical values from web.xml, in this case ServletContext concept will works great, i mean all servlets in the current web application can access these context values from the web.xml but its not the case in ServletConfig.

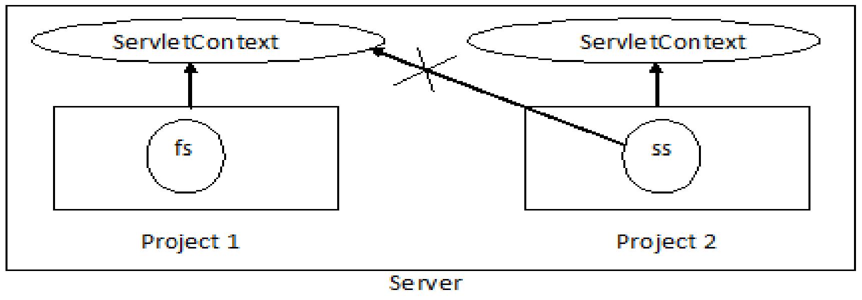


In a server we will have multiple ServletContext objects. This is based on number of projects available in the server. We can store the data into ServletContext object. The data stored in the ServletContext object can be accessible by all the resources of that project.



From the above diagram if we store the data into ServerContext object. FS and SS servlets can access the data.

If we store the data into one ServletContext object the other project can't access the data from current project.



How to Get ServletContext Object in Our Servlet Class

In servlet programming we have 3 approaches for obtaining an object of ServletContext interface

Way 1.

```
ServletConfig conf = getServletConfig();
ServletContext context = conf.getServletContext();
```

First obtain an object of ServletConfig interface ServletConfig interface contain direct method to get Context object, getServletContext();.

Way 2.
Direct approach, just call getServletContext() method available in GenericServlet [pre-defined]. In general we are extending our class with HttpServlet, but we know HttpServlet is the sub class of GenericServlet.

```
public class LoginServlet extends HttpServlet
{
public void doGet/doPost(-,-)
{
//�.
}
        ServletContext ctx = getServletContext();
}
```

Way 3.
We can get the object of ServletContext by making use of HttpServletRequest object, we have direct method in HttpServletRequest interface.

```
public class LoginServlet extends HttpServlet
{
public void doGet/doPost(HttpServletRequest req,-)
{
        ServletContext ctx = req.getServletContext();
}
}
```

How to Retrieve Data from ServletContext Interface Object

ServletContext provide these 2 methods, In order to retrieve the data from the web.xml [In web.xml we have write <context-param> tag to provide the values, and this <context-param> should write outside of <servlet> tag as context should be accessed by all servlet classes].
In general database related properties will be written in this type of situation, where every servlet should access the same data.

```
public String getInitParameter("param name");
public Enumeration getInitParameterNames();
```

ServletConfig

An object of ServletConfig is created by the web container for each servlet using its initialization phase. This object can be used to get configuration information from web.xml file.

Advantage of ServletConfig

If the configuration information is modified from the web.xml file, we don't need to change the Servlet. So it is easier to manage the web application if any specific content is modified from time to time.
The core advantage of ServletConfig is that you don't need to edit the Servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

How to get the ServletConfig object?

An object of ServletConfig can be obtained in 2 ways, they are…

Way 1

```
ServletConfig conf = getServletConfig();
```

In the above statement, we are directly calling getServletConfig() method as it is available in Servlet interface, inherited into GenericServlet and defined and further inherited into HttpServlet and later inherited into our own servlet class.

Way 2

```
ServletConfig object will be available in init() method of the servlet.
  public void init(ServletConfig config)
{
  // ......
}
```

So finally we are able to create ServletConfig object in our servlet class, then how to get the data from that object...?

How to Retrieve Data from ServletConfig Object

In order to retrieve the data of the ServletConfig we have two methods, which are present in ServletConfig interface.

```
public String getInitParameter("param name");
public Enumeration getInitParameterNames();
```

I am not going to explain about these methods, these are similar to 'Retrieve Client Input Data in Servlet' but here we are retrieving values from web.xml that's it.

In web.xml – <context-param> tag will be appear under <web-app> tag

So finally…….

No. of web applications = those many number of ServletContext objects [1 per web application]
No. of servlet classes = those many number of ServletConfig objects

Request Dispatcher interface

RequestDispacher is an interface that provides the facility to forward a request to another resource or include the content of another resource. RequestDispacher provides a way to call another resource from a servlet. Another resource can be servlet, jsp or html.

Methods of RequestDispacher interface:

1. forward(ServletRequest request,ServletResponse response): This method forwards a request from a servlet to another resource on the server.

Syntax:public void forward(ServletRequest request,ServletResponse response)throws ServletException, IOException

2. include(ServletRequest request,ServletResponse response): This method includes the content of a resource in the response.

Syntax: public void include(ServletRequest request,ServletResponse response)throws ServletException, IOException

How to get an object of RequestDispacher

RequestDispacher object can be gets from HttpServletRequest object.ServletRequest's getRequestDispatcher()method is used to get RequestDispatcher object.

RequestDispatcher requestDispatcher = request.getRequestDispatcher("/another resource");

After creating RequestDispatcher object you call forword or include method as per your requirement.

requestDispatcher.forward(request, response);

or

requestDispatcher.include(request, response);

Example:

Create LoginServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class LoginServlet extends HttpServlet {

	private static final long serialVersionUID = 8796051226600816058L;

	@Override
	protected void doPost(HttpServletRequest req, HttpServletResponse res)
			throws ServletException, IOException {

		PrintWriter out = res.getWriter();

		//Capturing the data from form
		String uname = req.getParameter("uname");
		String pwd = req.getParameter("pwd");

		if (uname.equals("admin") && pwd.equals("admin@123")) {
			RequestDispatcher rd = req.getRequestDispatcher("success.jsp");
			rd.forward(req, res);
		} else {
			out.println("<font color='red'><b>Invalid credentials</b></font>");
			RequestDispatcher rd = req.getRequestDispatcher("login.html");
			rd.include(req, res);
		}

	}

}
```

Create Login.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Page</title>
</head>
<body>
	<form action="login" method="post">
		Username:<input type="text" name="userName" /> <br /> <br />
		Password:<input type="password" name="password" /> <br /> <br />
		<input	type="submit" value="Login" />
	</form>
</body>
</html>
```

Create Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
            xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
            xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
            id="WebApp_ID" version="2.5">
            <display-name>ReqDispatchDemo</display-name>
            <welcome-file-list>
                        <welcome-file>login.html</welcome-file>
            </welcome-file-list>

            <servlet>
                        <servlet-name>loginServlet</servlet-name>
                        <servlet-class>LoginServlet</servlet-class>
            </servlet>

            <servlet-mapping>
                        <servlet-name>loginServlet</servlet-name>
                        <url-pattern>/login</url-pattern>
            </servlet-mapping>

    </web-app>
```

sendRedirect in servlet

sendRedirect () is the method of HttpServletResponse interface which is used to redirect response to another resource.

Syntax: response. sendRedirect(relative url);

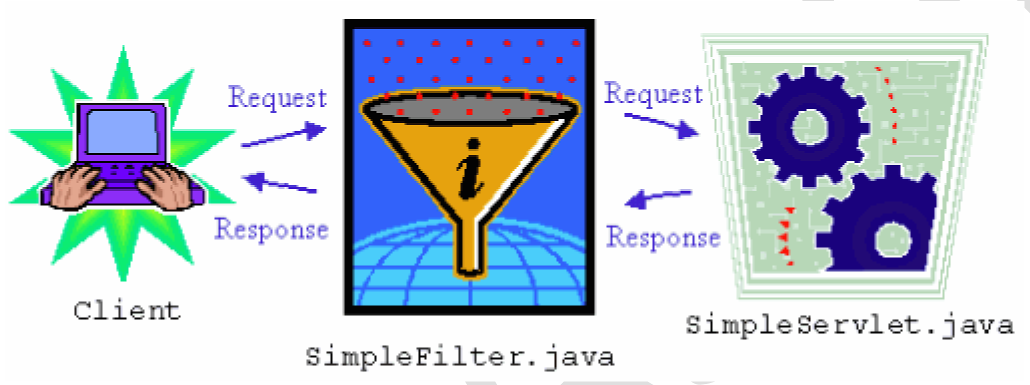
Difference between sendRedirect and RequestDispatcher.

sendRedirect	RequestDispatcher
1. Creates a new request from the client browser for the resource.	1. No new request is created.
2. Accept relative url so control can go inside or outside the server.	2. Not accept relative url so can go only inside the server.
3. New url can be seen in browser.	3. New url can't be seen in browser.
4. Work on response object.	4. Work on request object.

Servlet Filters

Servlet filters are powerful tools that are available to web application developers using the Servlet 2.3 specification or above. Filters are designed to be able to manipulate a request or response (or both) that is sent to a web application, yet provide this functionality in a method that won't affect servlets and JSPs being used by the web application unless that is the desired effect. A good way to think of servlet filters is as a chain of steps that a request and response must go through before reaching a servlet, JSP, or static resource such as an HTML page in a web application.

Definition: A filter is an object that performs filtering tasks either on the request to a resource, or on the response from a resource, or both.



Filters are used for several purpose which could be:

Authentication filters.
Logging and auditing filters.
Data compression filters.
Encryption filters.
Tokenizing Filters.
Filters that trigger resource access events.
When encapsulating common functionality in a filter, the same filter can easily be used with several servlets.

The Filter API

The Filter API consists of three interfaces:

javax.servlet.Filter
javax.servlet.FilterChain
javax.servlet.FilterConfig

The Filter Interface: There are three methods those need to be implemented in creating our own filter. According to the javadocs,

Init method: Called by the web container to indicate to a filter that it is being placed into service. We can write code in init method to get a reference to FilterConfig object which is like ServletConfig object and can read initialization parameters declared in web.xml.

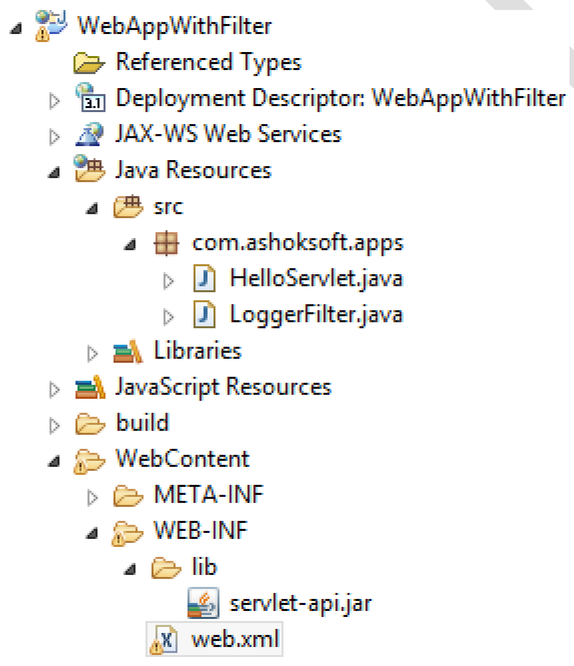
doFilter method : The doFilter method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain. The FilterChain passed in to this method allows the Filter to pass on the request and response to the next entity in the chain.

All our business logic for what filter is created goes into this method. For example, you can place log statements so that we can know after the resource is served and again filter is called after serving the resource.

destroy method: Called by the web container to indicate to a filter that it is being taken out of service.

Example on Configuring Filter in webapplciation.

Step 1: Create a web application like below



```
WebAppWithFilter
    Referenced Types
    Deployment Descriptor: WebAppWithFilter
    JAX-WS Web Services
    Java Resources
        src
            com.ashoksoft.apps
                HelloServlet.java
                LoggerFilter.java
        Libraries
    JavaScript Resources
    build
    WebContent
        META-INF
        WEB-INF
            lib
                servlet-api.jar
            web.xml
```

Step 2: create a HelloServlet which prints a message

---HelloServlet.java--

```
package com.ashoksoft.apps;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		PrintWriter out = response.getWriter();
		out.println("We are in HelloServlet");
		System.out.println("We are in HelloServlet");
	}
}
```

Step 3: Create logger filter

---LoggerFilter.java---

```
package com.ashoksoft.apps;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class LoggerFilter implements Filter {

	public void destroy() {
		// TODO Auto-generated method stub
	}

	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
			throws IOException, ServletException {
		System.out.println("Before visiting servlet");
		chain.doFilter(request, response);
		System.out.println("After visiting servlet");
	}

	public void init(FilterConfig fConfig) throws ServletException {
		// TODO Auto-generated method stub
	}
}
```

Step 4 : Configure filters and servlets in deployment descriptor (web.xml)

---web.xml---

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
```

```
		<display-name>WebAppWithFilter</display-name>
		<servlet>
			<servlet-name>HelloServlet</servlet-name>
			<servlet-class>com.ashoksoft.apps.HelloServlet</servlet-class>
		</servlet>
		<servlet-mapping>
			<servlet-name>HelloServlet</servlet-name>
			<url-pattern>/HelloServlet</url-pattern>
		</servlet-mapping>
		<filter>
			<filter-name>LoggerFilter</filter-name>
			<filter-class>com.ashoksoft.apps.LoggerFilter</filter-class>
		</filter>
		<filter-mapping>
			<filter-name>LoggerFilter</filter-name>
			<url-pattern>/ HelloServlet </url-pattern>
		</filter-mapping>
	</web-app>
```

--0--

Here notice that, the LoggerFilter is mapped to HelloServlet in filter-mapping tag. That means this filter is applied whenever a request for a servlet with name HelloServlet is made.

Step 5: Deploy and send request to Servlet

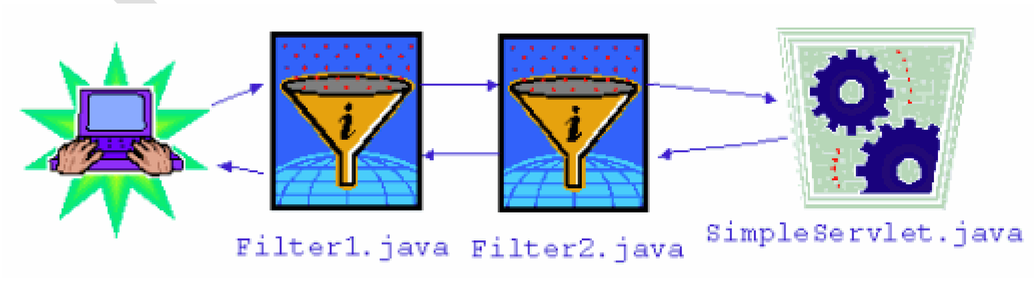
Deploy your application under tomcat server and hit url http://localhost:8080/WebAppWithFilter/HelloServlet .

You will see the messages as follows

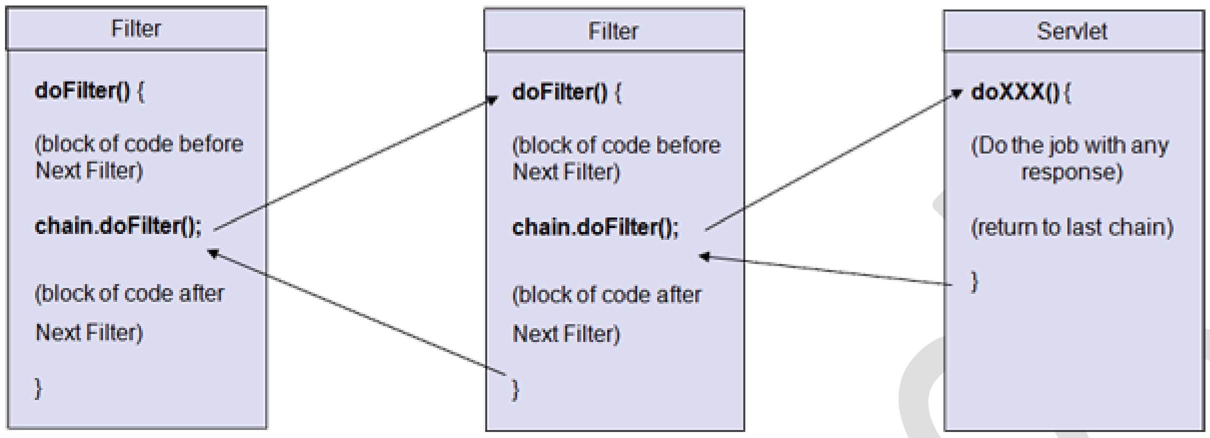
```
Before visiting servlet
We are in HelloServlet
After visiting servlet
```

Filter Chaining

A FilterChain is an object provided by the servlet container to the developer giving a view into the invocation chain of a filtered request for a resource. Filters use the FilterChain to invoke the next filter in the chain, or if the calling filter is the last filter in the chain, to invoke the resource at the end of the chain.



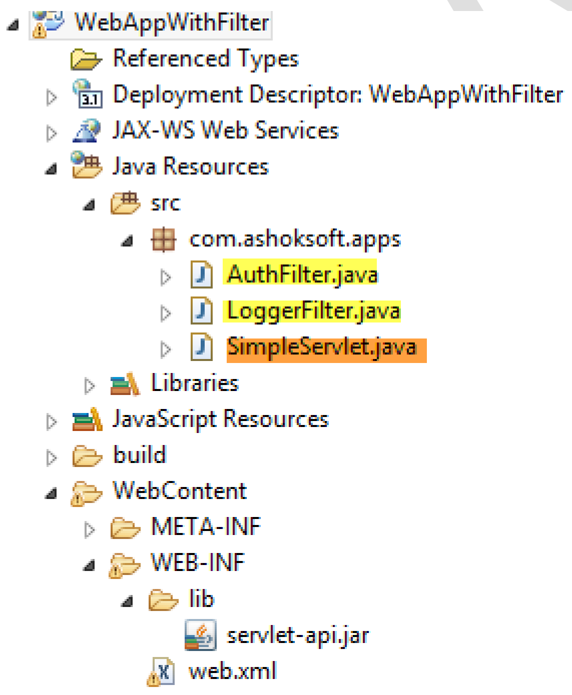
The javax.servlet.FilterChain interface is an argument to the doFilter() method. This interface has only one method, doFilter(), which are used by the programnmer to causes the next filter in the chain to be invoked. Methods in javax.servlet.FilterChain interface:


Filter
doFilter() {
(block of code before Next Filter)
chain.doFilter();
(block of code after Next Filter)
}
Filter
doFilter() {
(block of code before Next Filter)
chain.doFilter();
(block of code after Next Filter)
}
Servlet
doXXX() {
(Do the job with any response)
(return to last chain)
}

Note: The deployment descriptor is used to tell the container which, if any, filters in which sequence to call for each servlet in the application.

Example of Web application with Two Filters, to demonstrate on Filter Chaining

Step 1: Create a project



```
WebAppWithFilter
    Referenced Types
    Deployment Descriptor: WebAppWithFilter
    JAX-WS Web Services
    Java Resources
        src
            com.ashoksoft.apps
                AuthFilter.java
                LoggerFilter.java
                SimpleServlet.java
        Libraries
    JavaScript Resources
    build
    WebContent
        META-INF
        WEB-INF
            lib
                servlet-api.jar
            web.xml
```

Step 2: create a HelloServlet which prints a message

---HelloServlet.java--

```
package com.ashoksoft.apps;
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		PrintWriter out = response.getWriter();
		out.println("We are in HelloServlet");
		System.out.println("We are in HelloServlet");
	}
}
```

Step 3: Create logger & Authentication filter

--LoggerFilter.java--

```
package com.ashoksoft.apps;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class LoggerFilter implements Filter {

	public void destroy() {
		// TODO Auto-generated method stub
	}

	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
			throws IOException, ServletException {
		System.out.println("In Logger Filter before doFilter ");
		chain.doFilter(request, response);
		System.out.println("In Logger Filter after doFilter ");
	}

	public void init(FilterConfig fConfig) throws ServletException {
		// TODO Auto-generated method stub
	}
}
```

--AuthFilter.java---

```
package com.ashoksoft.apps;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class AuthFilter implements Filter {

	public AuthFilter() {
		// TODO Auto-generated constructor stub
	}

	public void destroy() {
		// TODO Auto-generated method stub
	}

	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
			throws IOException, ServletException {
		System.out.println("In AuthFilter before doFilter");
		chain.doFilter(request, response);
		System.out.println("In AuthFilter after doFilter");
	}

	public void init(FilterConfig fConfig) throws ServletException {
		// TODO Auto-generated method stub
	}

}
```

Step 4 : Configure Servlet & Filter in deployment descriptor (web.xml)

---web.xml---

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
	<servlet>
		<servlet-name>HelloServlet</servlet-name>
		<servlet-class>com.ashoksoft.apps.HelloServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>HelloServlet</servlet-name>
		<url-pattern>/HelloServlet</url-pattern>
	</servlet-mapping>
	<filter>
		<filter-name>LoggerFilter</filter-name>
		<filter-class>com.ashoksoft.apps.LoggerFilter</filter-class>
	</filter>
	<filter-mapping>
```

```
                    <filter-name>LoggerFilter</filter-name>
                    <url-pattern>/HelloServlet</url-pattern>
            </filter-mapping>
            <filter>
                    <filter-name>AuthFilter</filter-name>
                    <filter-class>com.ashoksoft.apps.AuthFilter</filter-class>
            </filter>
            <filter-mapping>
                    <filter-name>AuthFilter</filter-name>
                    <url-pattern>/HelloServlet</url-pattern>
            </filter-mapping>
    </web-app>
```

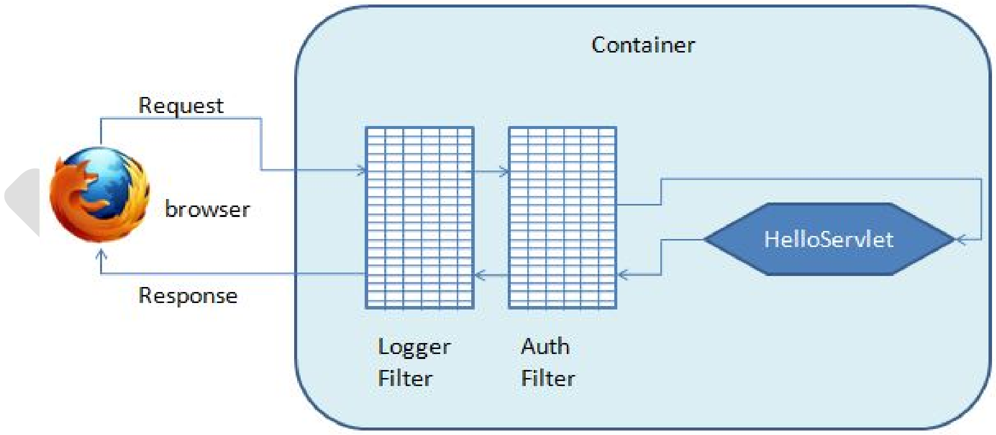
---0--

Step 5: Deploy and send request to Servlet

Deploy your application under tomcat server and hit url http://localhost:8080/WebAppWithFilter/HelloServlet .You will see the messages as follows

```
In LoggerFilter before doFilter
We are in HelloServlet
In LoggerFilter after doFilter
In AuthFilter after doFilter
```

What is happening here?
We see that both filters are called. But what decides the filter order? Who knows about which filters are to be placed in sequence? What is the significance of FilterChain in filter order?



- In web.xml , we have declared two filters i.e. LoggerFilter and AuthFilter
- LoggerFilter comes first in web.xml
- AuthFilter is declared after LoggerFilter .
- When filters are registered, they are registered according to the order specified in web.xml.
- FilterChain object knows what filter or resource to be called when doFilter method is called from a Filter.
- So in this case, as AuthFilter comes after LoggerFilter , AuthFilter is called after LoggerFilter .

- After AuthFilter is called, HelloServlet is called and when its doGet method completes, again control goes back to AuthFilter to complete the execution.
- Again after AuthFilter is completed, LoggerFilter is called.
- Now what if we want to change the filter order so that AuthFilter is called first and then LoggerFilter , we simply need to change the filter order in web.xml and FilterChain object will do the rest of the work.

If there are two filters, for example, the key steps of this mechanism would be as follows

- The target servlet is requested. The container detects that there are two filters and creates the filter chain.
- The first filter in the chain is invoked by its doFilter() method.
- The first filter completes any preprocessing, then calls the doFilter() method of the filter chain. This results in the second filter being invoked by its doFilter() method.
- The second filter completes any preprocessing, then calls the doFilter() method of the filter chain. This results in the target servlet being invoked by its service() method.
- When the target servlet is finished, the chain doFilter() call in the second filter returns, and the second filter can do any postprocessing.
- When the second filter is finished, the chain doFilter() call in the first filter returns, and the first filter can do any postprocessing.
- When the first filter is finished, execution is complete.
- None of the filters are aware of their order. Ordering is handled entirely through the filter chain, according to the order in which filters are configured in web.xml.

Servlet Listeners

Listeners are the classes which listens to a particular type of events and when that event occurs , triggers the functionality. Each type of listener is bind to a type of event. In this chapter we will discuss the types of listeners supported by servlet framework.

During the lifetime of a typical web application, a number of events take place, such as:

- requests are created or destroyed,
- request or session attributes are added, removed, or modified, and so on and so forth.

The Servlet API provides a number of listener interfaces we can implement in order to react to these events. There are total eight type of listeners available in servlet framework which listens to a particular event and they are –

1. **ServletRequestListener**
2. **ServletRequestAttributeListener**
3. **ServletContextListener**
4. **ServletContextAttributeListener**
5. **HttpSessionListener**
6. **HttpSessionAttributeListener**
7. **HttpSessionActivationListener**
8. **HttpSessionBindingListener**

As the configurations of servlets, filters goes inside web.xml, similarly listeners are also configured inside web.xml using <listener> </listener> tag.

```
Syntax:                 <listener>
                             <listener-class></listerner-class>
                        </listener>
```

Note – Listeners are neither a Servlets nor a JSP.

ServletRequestListener

ServletRequestListener listens to ServletRequestEvent event which gives a notification when request is created or destroyed.

What is the need?
We may like to receive a notification whenever a request for a resource is made from the client so that we can log it.

How can we achieve it?
Java EE specification provides an interface called ServletRequestListener which receives notifications whenever a new request is about to come to the web application. This is what javadoc says about it.

A ServletRequest is defined as coming into scope of a web application when it is about to enter the first servlet or filter of the web application, and as going out of scope as it exits the last servlet or the first filter in the chain.

- **requestInitialized (ServletRequestEvent event):** Receives notification that a ServletRequest is about to come into scope of the web application.
- **requestDestroyed (ServletRequestEvent event) :** Receives notification that a ServletRequest is about to go out of scope of the web application.

Example:

```
-----------------------------------------------------MyServletRequestListener.java--------------------------------------------
package com.ashoksoft.apps;
import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;
public class MyServletRequestListener  implements ServletRequestListener {
   /**
    * Default constructor.
    */
   public MyServletRequestListener() {
   }
   public void requestDestroyed(ServletRequestEvent servletRequestEvent)
   {
      ServletRequest request = servletRequestEvent.getServletRequest();
      System.out.println("Request Destroyed");
   }
   public void requestInitialized(ServletRequestEvent servletRequestEvent)
   {
      ServletRequest request = servletRequestEvent.getServletRequest();
      System.out.println("Request initialized");
    }
}
```

--web.xml--

```
<listener>
 <description> Servlet Request Listener Example</description>
 <listener-class> com.ashoksoft.apps.MyServletRequestListener</listener-class>
</listener>
```

ServletRequestAttributeListener

ServletRequestAttributeListener listens to ServletRequestAttributeEvent event which gives a notification when any object is added, removed or replaced from request .

ServletRequestAttributeListener is the interface and it defines three methods –

- **attributeAdded(ServletRequestAttributeEvent e):** It notifies whenever a new attribute is added to the request.
- **attributeRemoved(ServletRequestAttributeEvent e):** It notifies whenever the attribute is removed from the request.
- **attributeReplaced(ServletRequestAttributeEvent e):** It notifies whenever the attribute gets replaced on the request.

Attribute name and value that has been added, removed or replaced can be obtained from the getName() and getValue() method of ServletRequestAttributeEvent

ServletContextListener

ServletContextEvent class gives notifications about changes to the servlet context of a web application. ServletContextListener receives the notifications about changes to the servlet context and perform some action. ServletContextListener is used to perform important task at the time when context is initialized and destroyed. In short, ServletContextEvent and ServletContextListener works in pair, whenever Servlet COntext changes, ServletContextEvent publishes a notification which is received by ServletContextListener and then, based on that certain tasks are performed by it.

ServletContextListener is the interface and it defines two methods –

- **void contextDestroyed(ServletContextEvent e) –** This method is executed when application is destroyed
- **void contextInitialized(ServletContextEvent e)-** This method is executed when application is initialized

ServletContext object can be obtained from ServletContextEvent and listener can set the attributes in Servlet context object which can be used in servlets later.
We can use the "ServletContextListener " listener for any activity that is required either at the application deployment time or any clean up activity required when application is destroyed. One of the practical example that I can think of is initializing database connections and clean up of database connections.

ServletContextAttributeListener

ServletContextAttributeListener listens to SessionContexAttributetEvent event which gives a notification when any object is added, removed or replaced from servlet context .

ServletContextAttributeListener is the interface and it defines three methods –

- **attributeAdded(ServletContextAttributeEvent e):** It notifies whenever a new attribute is added to the servlet context.
- **attributeRemoved(ServletContextAttributeEvent e):** It notifies whenever the attribute is removed from the servlet context.
- **attributeReplaced(ServletContextAttributeEvent e):** It notifies whenever the attribute gets replaced on the servlet context.

HttpSessionListener

HttpSessionListener listens to HttpSessionEvent event which gives a notification when session is created or destroyed.

HttpSessionListener is the interface and it defines two methods –

- **void sessionDestroyed(HttpSessionEvent e) –** This method is executed when session is destroyed
- **void sessionCreated (HttpSessionEvent e) -** This method is executed when session is created. Session object can be obtained from HttpSessionEvent.

HttpSessionAttributeListener

HttpSessionAttributeListener listens to HttpSessionBindingEvent event which gives a notification when any object is added, removed or replaced from session.

HttpSessionAttributeListener is the interface and it defines three methods –

- **attributeAdded (HttpSessionBindingEvent e):** It notifies whenever a new attribute is added to the session.
- **attributeRemoved (HttpSessionBindingEvent e):** It notifies whenever the attribute is removed from the session.
- **attributeReplaced (HttpSessionBindingEvent e):** It notifies whenever the attribute gets replaced on the session.

Annotations

Java Annotations allow us to add metadata information into our source code, although they are not a part of the program itself. Annotations were added to the java from JDK 5. Annotation has no direct effect on the operation of the code they annotate (i.e. it does not affect the execution of the program).

What's the use of Annotations?

1) Instructions to the compiler: There are three built-in annotations available in Java (@Deprecated, @Override & @SuppressWarnings) that can be used for giving certain instructions to the compiler. For example the @override annotation is used for instructing compiler that the annotated method is overriding the method. More about these built-in annotations with example is discussed in the next sections of this article.

2) Compile-time instructors: Annotations can provide compile-time instructions to the compiler that can be further used by sofware build tools for generating code, XML files etc.

3) Runtime instructions: We can define annotations to be available at runtime which we can access using java reflection and can be used to give instructions to the program at runtime. We will discuss this with the help of an example, later in this same post.

Annotations basics

An annotation always starts with the symbol @ followed by the annotation name. The symbol @ indicates to the compiler that this is an annotation.

For e.g. `@Override`

here @ symbol represents that this is an annotation and the Override is the name of this annotation.

Built-in Annotations in Java

Java has three built-in annotations:

- @Override
- @Deprecated
- @SuppressWarnings

1) @Override:

While overriding a method in the child class, we should use this annotation to mark that method. This makes code readable and avoid maintenance issues, such as: while changing the method signature of parent class, you must change the signature in child classes (where this annotation is being used) otherwise compiler would throw compilation error. This is difficult to trace when you haven't used this annotation.

```
public class MyParentClass {

   public void justaMethod() {
      System.out.println("Parent class method");
   }
}
public class MyChildClass extends MyParentClass {

   @Override
   public void justaMethod() {
      System.out.println("Child class method");
   }
}
```

2) @Deprecated

@Deprecated annotation indicates that the marked element (class, method or field) is deprecated and should no longer be used. The compiler generates a warning whenever a program uses a method, class, or field that has already been marked with the @Deprecated annotation. When an element is deprecated, it should also be documented using the Javadoc @deprecated tag, as shown in the following example. Make a note of case difference with @Deprecated and @deprecated. @deprecated is used for documentation purpose.

Example:

```
@Deprecated
public void anyMethodHere(){
   // Do something
}
```

3) @SuppressWarnings

This annotation instructs compiler to ignore specific warnings. For example in the below code, I am calling a deprecated method (lets assume that the method deprecatedMethod() is marked with @Deprecated annotation) so the compiler should generate a warning, however I am using @@SuppressWarnings annotation that would suppress that deprecation warning.

```
@SuppressWarnings("deprecation")
  void myMethod() {
    myObject.deprecatedMethod();
}
```

Servlets with Annotation (feature of servlet3):

The Servlet API 3.0 introduces a new package called javax.servlet.annotation which provides annotation types which can be used for annotating a servlet class. The annotations can replace equivalent XML configuration in the web deployment descriptor file (web.xml) such as servlet declaration and servlet mapping. Servlet containers will process the annotated classes at deployment time. The annotation types introduced in Servlet 3.0 are:

1. **@HandlesTypes**
2. **@HttpConstraint**
3. **@HttpMethodConstraint**
4. **@MultipartConfig**
5. **@ServletSecurity**
6. **@WebFilter**
7. **@WebInitParam**
8. **@WebListener**
9. **@WebServlet**

@WebServlet: @WebServlet annotation is the replacement of servlet configuration in web.xml. When we annotate our servlet class with @WebServlet annotation the container will be able to recognize this as a servlet at the loading time.

Class annotated with @WebServlet still needs to extend the HttpServlet class. With this annotation we can specify servlet-name, url-mapping, load on Start up, description, init params, async supported etc

Let’s take an example Welcome Servlet which is defined in a web.xml like below

```
<servlet>
  <description>Welcome Servlet</description>
   <servlet-name>WelcomeServlet</servlet-name>
   <servlet-class>com.servlet.tutorial.WelcomeServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
<servlet-mapping>
   <servlet-name>WelcomeServlet</servlet-name>
   <url-pattern>/WelcomeServlet</url-pattern>
</servlet-mapping>
```

Same servlet configuration can be done with **@WebServlet** annotation as shown below. @WebServlet annotation is highlighted below and all the attributes configured in web, xml are configured in annotation itself. With this approach, we need not to configure any entry in web.xml for Welcome Servlet.

Attributes of @WebServlet annotation

- **name- defines the name of servlet**
- **urlPatterns – maps the servlet to the pattern**
- **loadOnStartup – defines the value of load on start up**
- **description –description about servlet**
- **initParams – takes multiple @WebInitParam annotation**

---**AnnotationServlet.java**--

```
package info.ashok;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name="/AnnotatedServlet",urlPatterns="/AnnotatedServlet ",loadOnStartup=1,
description="Welcome Servlet")
public class AnnotationServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request,
				HttpServletResponse response) throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
		out.print("<h1>Hello World example using annotations.</h1>");
		out.close();
	}
 }
```

--0--

Note: Did you notice the word “urlPatterns”? It is not a pattern, it is patterns which means we can map a servlet with multiple URLs like below

Syntax :

```
@WebServlet (
		name="/AnnotatedServlet",
		urlPatterns= {"/AnnotatedServlet","/HelloServlet"},
		loadOnStartup=1,
		description="Welcome Servlet"
  )
```

@WebInitParam: This annotation is used for init-param configurations of servlet. Remember we can define multiple init parameters for a servlet which we can get in the servlet using servlet config object.

Attributes of @WebInitParam annotation

Name – name of init param
Value – value of init param
Description –description of init param

One @WebInitParam annotation is needed for one init param tag and all @WebInitParam annotation are configured in initParams attribute of @WebServlet annotation.

Syntax :

```
@WebServlet(
        urlPatterns="/WelcomeServlet",
        description="My Welcome Servlet",
        initParams={
                @WebInitParam(name="param1",value="value 1",description=" param 1 description "),
                @WebInitParam(name="param2", value="value 2", description=" param 2 description ")
        }
 })
```

@WebFilter : @WebFilter annotation is the replacement of filter configuration in web.xml. When we annotate our filter class with @WebFilter annotation the container will be able to recognize this as a filter at the loading time. With this annotation we can specify filter-name, url-mapping, description ,init params ,async supported etc.

Attributes for @WebFilter annotation

filterName- defines the name of filter
urlPatterns – maps the servlet to the pattern
description –description about servlet
initParams – takes multiple @WebInitParam annotation

Syntax :

```
@WebFilter(
        urlPatterns="/*",
        description="Logging Filter",
        initParams={
                @WebInitParam(name="param1",value="value 1"),
                @WebInitParam(name="param2", value="value 2")
        }
})
```

@MultipartConfig : With Servlet 3.0 , there is an inbuilt support for File Upload.When this annotation is added on the servlet, container knows that the Request this servlet is expecting will have multipart/form-data MIME data. This annotation can specify location to store the file on server, maxFileSize allowed for uploaded files, maxRequestSize allowed for multipart/form-data requets, and fileSizeThreshold after exceeding it the file content will be written on the disk.

@WebListener : @WebListener annotation is the replacement of listener configuration in web.xml. When we annotate our listener class with @WebListener annotation the container will be able to recognize this as a listener at the loading time. The class still needs to extend the appropriate listener class.

JSP

JSP (Java Server Pages) is Oracle specification and a server side technology used to implement presentation part of web application. JSP runs on the server machine and capable of rendering dynamic views as compared to HTML which can render static content only.

JSP technology is used to create dynamic web application same like Servlet technology. It is another web technology given by Sun MicroSystem for the development of dynamic web pages an the client browser. It provides a tag based approach to develop java web components.

JSP have .jsp extension, we can directly access these JSP pages from client system browser window. Because jsp pages are contains outside of the WEB-INF folder.

A JSP page consists of Html tags and JSP tags. The jsp pages are easier to maintain than servlet. It provides some additional features such as Expression Language, Custom Tag etc.

A JSP is called as page but not program because a JSP contains totally tags. Every JSP is internally converted into a Servlet by the server container.

Why JSP?

The first technology given for the development of web application is CGI. In CGI have some drawback, So Sun MicroSystem develop a new technology is servlet. But working with Servlet Sun MicroSystem identify some problem, Servlet technology need in depth java code and Servlet is failed to attract the programmer.

To overcome the problem with Servlet technology we use jsp technology.

JSP Tag

A JSP page contains both html tags and JSP tags. Html tags will produce static output and JSP tags will produced dynamic output. Because of JSP tags we called as JSP is dynamic web page.

JSP

JSP is the extension of servlet that provides the solution of all above problems. JSP defines a structure which is java code inside HTML code. With this structure it simplifies the web development.

Javax.servlet.jsp package.

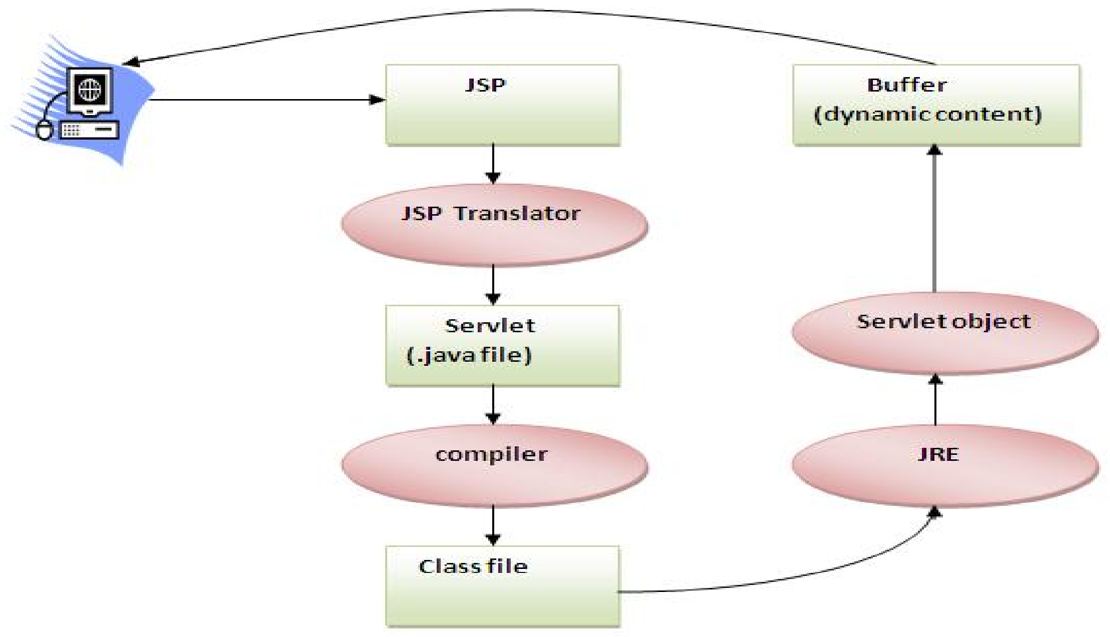
Javax.servlet.jsp and its sub packages provide the classes and interfaces facilitate the development of JSP.

How JSP page works?

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.

For first request all 6 steps will be executed but for next requests of this JSP page 2nd and 3rd step will not execute.



Difference Between JSP and HTML

Html is a Client side Technology and JSP is a Server side Technology. Some other differences are given below;

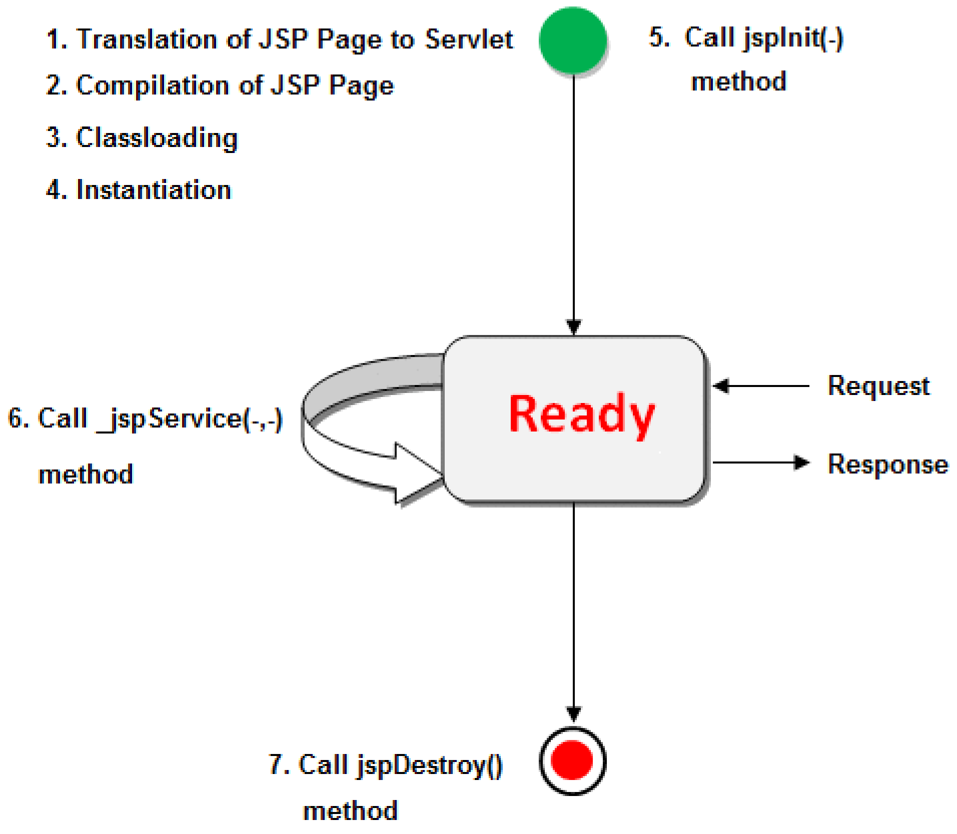
	HTML	JSP
1	Html is given by w3c (World Wide Web Consortium).	JSP is given by SunMicro System.
2	Html generated static web pages.	JSP generated dynamic web pages.
3	It do not allow to place java code inside Html pages.	JSP allows to place java code inside JSP pages.
4	It is Client side technology	It is a Server side technology.
5	Need Html Interpreter to execute these code.	Need JSP container to execute jsp code.
6	It does not allow to place custom tag or third party tag.	It allow to place custom tag or third party tag.

JSP Life Cycle

A **JSP life cycle** can be defined as the entire process from its creation till the destruction which is similar to a Servlet life cycle with an additional step which is required to translate a JSP into Servlet.

Life cycle of a JSP Page

- Translation of JSP Page to Servlet
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).


1. Translation of JSP Page to Servlet
2. Compilation of JSP Page
3. Classloading
4. Instantiation
5. Call jspInit(-) method
Ready
Request
Response
6. Call _jspService(-,-) method
7. Call jspDestroy() method

Life cycle method of JSP

- jspInit()
- _jspService()
- jspDestroy()

We can override jspInti(), jspDestroy(). But we can not override _jspService() method.

To inform the programmer that you should not override the service() method the method name is started with '_' symbol.

Features of JSP

JSP are tag based approach to develop dynamic web application. JSP have all the features of Servlet because it is internally Servlet. It have following feature these are;

- Extension to Servlet
- Powerful
- Portable
- Flexible
- Easy
- Less code than Servlet

Extension to Servlet: JSP is Extension to Servlet, it have all the features of servlet and it have also implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP easy to develop any application.
Powerful: These are internally Servlet, means consists byte code, so that all java features are applicable in case of jsp like robust, dynamic, secure, platform independent.
Portable: JSP tags will process and execute by the server side web container, So that these are browser independent and j2ee server independent.
Flexible: Allows to defined custom tags, the developer can fill conferrable to use any kind, framework based markup tags in JSP.
Easy: JSP is easy to learn, easy to understand and easy to develop. JSPs are more convenient to write than Servlets because they allow you to embed Java code directly into your HTML pages, in case of servlets you embed HTML inside of Java code.
Less code than Servlet: In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code.

JSP Translation

Each JSP page is internally translated into an equivalent Servlet. Every JSP tag written in the page will be internally converted into an equivalent java code by the containers this process is called **translation**.

There are two phases are occurs in JSP;

- Translation phase
- Request processing phase

Translation phase: It is process of converting jsp into an equivalent Servlet and then generating class file of the Servlet.

Request processing phase: It is process of executing service() of jsp and generating response data on to the browser.

When first time a request to a jsp then translation phase occurs first and then after service phase will be executed. From next request to the jsp, only request processing phase is got executed but translation phase is not because jsp is already translated.

In following two cases only translation occurs;

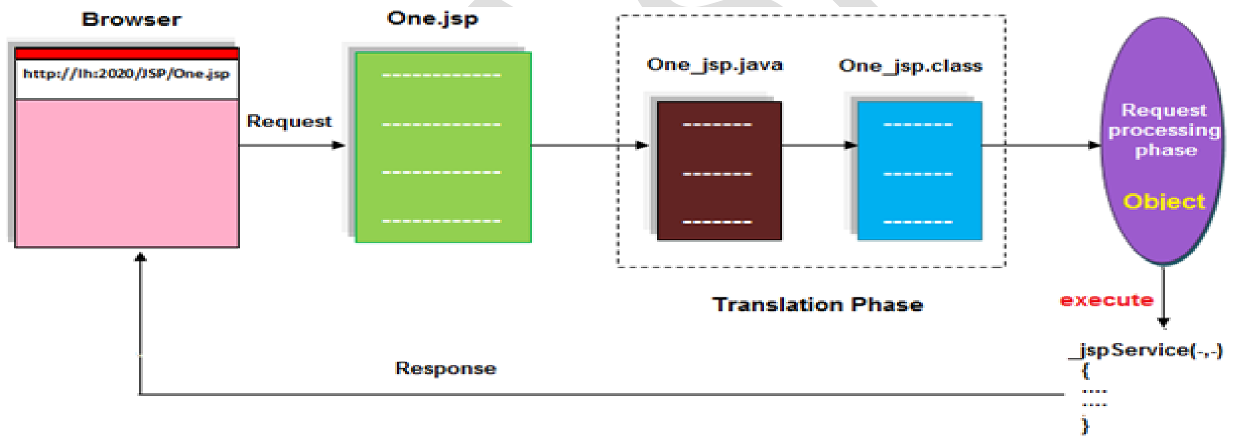
- When first request is a given to the jsp.
- When a jsp is modified.

If a request is given to the jsp after it has modified then again translation phase is executed first and after that request processing phase will be executed.

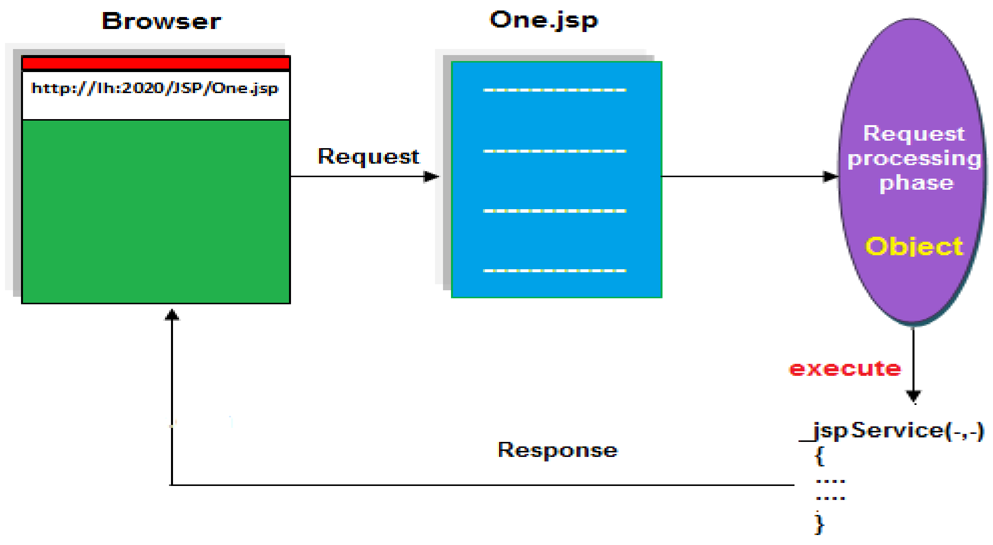
Note:

- If we shutdown and restart the Servet then only request processing phase will be executed, Because when we shutdown the server java and class files are not deleted from the server.
- If we remove the class file from server then partial translation occurs..
- If we remove both java and class file from server then again translation occurs.
- When a jsp is translated into an equivalent Servlet then that Servlet extends some base class provided by the server, but internally it extends HttpServlet.

When first request is come or modify jsp page.



From next request



Configuring JSP File

Configuring a JSP into a web.xml file is optional because JSP is a public file to the web application.
A JSP called a public file and servlet is called a private file of the web application. Because JSP files stored in root directory of the web application and Servlet class file stored in sub directory of the web application. Because JSP is the public file, we can directly send a request from a browser by using its file name.
If we want ot configure a JSP in web.xml file the the xml elements are same as Servlet-configuration. We need to replace <servlet-class> element with <jsp-file>

Syntax

```
<web-app>
<servlet>
<servlet-name>test</servlet-name>
<jsp-file>/One</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>test</>
<url-pattern>/srv1</url-pattern>
</servlet-mapping>
</web-app>
```

Note: If we configure a jsp in web.xml then we can send the request to the jsp either by using jsp filename or by using its url-pattern.

Example

```
httpp://localhost:2014/root/One.jsp
```

or

httpp://localhost:2014/root/srv1

When we need Servlet and JSP ?

For huge request processing logic with less response generation logic we need Servlet

With less request processing logic with more response generation logic we need JSP.

Problems with Servlet

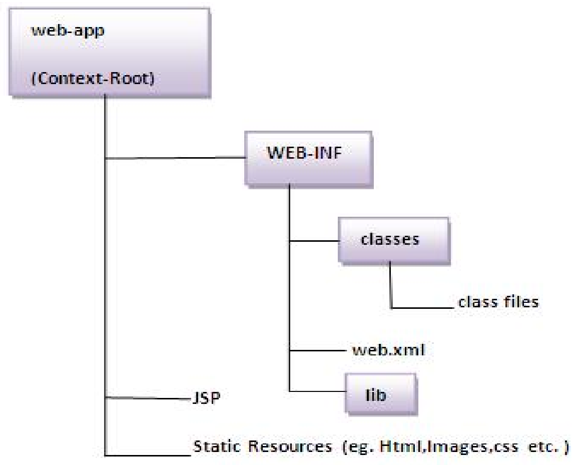
If any changes in static html code of Servlet, the entire Servlet need recompilation, redeployment, needs server restart. Any modification in Servlet needs recompilation because both request processing logic and response generation logic are tight coupled.

Difference Between Servlet And JSP

Servlet	**JSP**	
1	Servlet is faster than jsp	JSP is slower than Servlet because it first translate into java code then compile.
2	In Servlet, if we modify the code then we need recompilation, reloading, restarting the server> It means it is time consuming process.	In JSP, if we do any modifications then just we need to click on refresh button and recompilation, reloading, restart the server is not required.
3	Servlet is a java code.	JSP is tag based approach.
4	In Servlet, there is no such method for running JavaScript at client side.	In JSP, we can use the client side validations using running the JavaScript at client side.
5	To run a Servlet you have to make an entry of Servlet mapping into the deployment descriptor file i.e. web.xml file externally.	For running a JSP there is no need to make an entry of Servlet mapping into the web.xml file externally, you may or not make an entry for JSP file as welcome file list.
6	Coding of Servlet is harden than jsp.	Coding of jsp is easier than Servlet because it is tag based.
7	In MVC pattern, Servlet plays a controller role.	In MVC pattern, JSP is used for showing output data i.e. in MVC it is a view.
8	Servlet accept all protocol request.	JSP will accept only http protocol request.
9	In Servlet, aervice() method need to override.	In JSP no need to override service() method.
10	In Servlet, by default session management is not enabled we need to enable explicitly.	In JSP, session management is automatically enabled.

11	In Servlet we do not have implicit object. It means if we want to use an object then we need to get object explicitly form the servlet.	In JSP, we have implicit object support.
12	In Servlet, we need to implement business logic, presentation logic combined.	In JSP, we can separate the business logic from the presentation logic by uses javaBean technology.
13	In Servlet, all package must be imported on top of the servlet.	In JSP, package imported anywhere top, middle and bottom.

Directory Structure of JSP



Because of jsp page contains outside of WEB-INF, JSP is a public file to the web application. You can directly access these JSP pages from client system browser window.

Jsp First program

Here i will give you a simple jsp example. First you write your first jsp code on any editor and save with .jsp extension like below;

Save and compile jsp program

JSP program must be save with the **.jsp** extension. And for compile jsp code simply follow step of compilation servlet code.

```
<html>
```

```
<body>
<% out.print("Hello word"); %>
</body>
</html>
```

Result

```
Hello word
```

How to run jsp program

After write your jsp code, deploy jsp program in your root directory and start server. Follow below steps to run jsp code.

- Deploy jsp code in root directory
- Start Server
- Visit on browser or give request with url like below

Syntax

```
http://localhost:portnumber/RootDirectory/jspfile
```

Example

```
http://localhost:2015/jspapplication/index.jsp
```

Scripting Element

In JSP, java code can be written inside the jsp page using the scriptlet tag. JSP Scripting elementare written inside **<% %>** tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page.

Jsp scripting elements are classified into two types are;

- Language Based Scripting Element

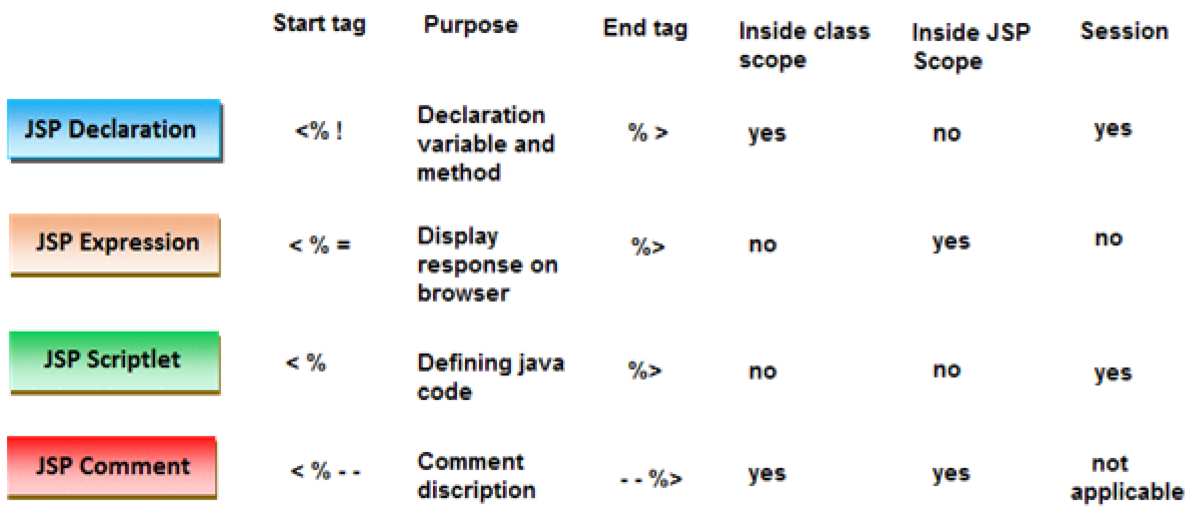
- Advance Scripting Elements (Expression Language)

Language Based Scripting Element

These are used to defined script in jsp page, this is traditional approach to define the script in jsp page.
Language Based Scripting Element are classified into 4 types, they are;

- Comment Tag
- Declaration Tag
- Expression Tag
- Scriptlet Tag

Scripting Element Detail



	Start tag	Purpose	End tag	Inside class scope	Inside JSP Scope	Session
JSP Declaration	<% !	Declaration variable and method	% >	yes	no	yes
JSP Expression	< % =	Display response on browser	%>	no	yes	no
JSP Scriptlet	< %	Defining java code	%>	no	no	yes
JSP Comment	< % - -	Comment discription	- - %>	yes	yes	not applicable

JSP Comment

This is used to define comment description in jsp page. In jsp page we can insert three types of comments they are;

- Jsp comment
- Html comment
- Java comment

JSP Comment

This type of comment is called as hidden comment, because it is invisible when a jsp is translated into a servlet internally.

Syntax

```
<%-- Comment discription %>
```

Html Comment

Html comment tag is common for Html, xml and jsp. In a jsp page if we write html comment these comment are visible in the _jspService(-,-) of the internal Servlet.

```
<!--  Comment discription -->
```

Java Comment

We can write a single or multiline comment of java in a scriptlet tag. Java comment are allowed only inside scriptlet tags because we can insert java code in a jsp only at a scriptlet tags.

```
//  Single line comment

/*
Multiline comment
*/
```

JSP Declaration Tag

This is used to declare variable and methods in jsp page will be translate and define as class scope declaration in .java file.

The variable and methods will become global to that jsp. It means in that jsp we can use those variables any where and we can call those method any where.

At the time of translation container inserts the declaration tag into the class. So the variables become instants variables and methods will become instants methods.

The code written inside the jsp declaration tag is placed outside the service() method of auto-generated Servlet, so it does not get memory at each request.

Syntax: <%! Variable or method Declaration %>

JSP Expression Tag

Expression tag is used, to find the result of the given expression and send that result back to the client browser. In other words; These are used to show or express the result or response on browser. We can use this as a display result of variable and invoking method.
Each Expression tag of jsp will be internally converted into **out.print()** statement. In jsp out is an implicit object.

The expression tags are converted into out.print() statement and insert into the _jspService(-,-) of the servlet class by the container.

The expression tags written in a jsp are executed for each request, because _jspService(-,-) is called for each request.

One expression tag should contains one java expression only. In a jsp we can use expression tag for any number of times.

```
<%= expression %>
```

Expression does not need any semicolon (;) by default it places under jsp service() method scope.

Note: In jsp, the implicit object are allowed into the tags, which are inserted into _jspService(-,-). An expression tag goes to _jspService(-,-) only. So implicit object are allowed in the expression tag.

JSP Scriptlet Tag

It use to define or insert java code in a jsp. Every java statement should followed by semicolon (;). It will be place into jspService() method.

The scriptlet tags goes to _jspService(-,-), during translation. It means scriptlet tags are executed for each request. Because _jspService(-,-) is called for each request.

We can use implicit object of jsp in a scriptlet tag also because scriptlet tag goes to _jspService(-,-) only. In a jsp implicit object are allowed in expression tags and scriptlet tags but not in the declaration tags.

```
<% java code %>
```

JSP directives

JSP directives provides the instructions and directions to the web container about how to control the processing JSP page.

The directive elements are used to do page related operation during page translation time. JSP supports 3 types of directive elements, they are;.

Types of JSP directives.

- Page directive
- Include directive
- Taglib directive

```
<@ directive attribute="value" %>
```

JSP page directive

Page directive is used to provide the instructions to the web container that are specific to the current JSP page. It defines the page specific attributes like scripting language, error page etc.

```
<@ page attribute="value" %>
```

Page directive attributes

- import
- contentType
- extends
- language
- buffer
- info
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

JSP include directive

JSP include directive is used to merge or include the content of other resource into the current JSP page during translation phase. Other resource can be jsp, html or a text file. It provides the facility of code reusability. This is known as static include because the target page is located and included at the time of page compilation.

Advantage of Include directive

Code Re-usability

Syntax of Include directive elements

```
<%@ include file="resourceName"%>
<html>
<body>

<%@ include file="footer.html" %>

</body>
</html>
```

Note: The include directive includes the original content, so the actual page size grows at run-time.

Taglib Directive

This is used to use custom tags in JSP page, here the custom tag may be user defined ot JSTL tag or strust, JSF,..... etc.

```
<@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Attributes of JSP taglib directive

1. uri: This attribute specify the location of the tag library.

2. prefix: This attribute is to inform the web container that this markup is used for custom actions.

JSP implicit objects

These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are being created inside service method so we can directly use them within **Scriptlet** without initializing and declaring them. There are total 9 implicit objects available in JSP.

Implicit Objects and their corresponding classes:

out	javax.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
exception	javax.servlet.jsp.JspException
page	java.lang.Object
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

out

It's an instance of **javax.servlet.jsp.JspWriter**. This allows us to access Servlet output stream. The output which needs to be sent to the client (browser) is passed through this object. In simple words out implicit object is used to write content to the client.

```
<%
out.print("Welcome");
%>
```

request

The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP we often prompts user to fill in those details, this object is then used to get those entered details on an another JSP page (action page) for validation and other purposes.Example of request implicit object where we are printing the name of the user with welcome message.

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
```

```
</form>
```

welcome.jsp

Example

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

response

It is basically used for modfying or delaing with the response which is being sent to the client(browser) after processing the request.

Example of response implicit object

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

config

config object is an implicit object of type ServletConfig and it is created by the container, whenever servlet object is created. This object can be used to get initialization parameter for a particular JSP page.

config object is created by the web container for every jsp page. It means if a web application has three jsp pages then three config object are created.

In jsp, it is not mandatory to configure in web.xml. If we configure a jsp in web.xml then the logical name and init parameter given in web.xml file are stored into config object by the container.

config object is accessible, only if the request is given to the jsp by using its url pattern, but not with name of the jsp.

Example of config implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
<servlet-name>Home</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>

<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>Home</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));
```

```
String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

page

In JSP, page is an implicit object of type Object class. When a jsp is translated to an internal Servlet, we can find the following statement in the service() method of servlet.

Object page=this;

For using this object it must be cast to Servlet type.For example:

```
<% (HttpServlet)page.log("message"); %>
```

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

Example

```
<% this.log("message"); %>
```

session

In JSP, session is an implicit object of type HttpSession.This object used to set,get or remove attribute or to get session information.

Example of Session implicit object

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

```
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>
```

One.jsp

```
<%

String name=(String)session.getAttribute("user");
out.print("Hello "+name);

%>
```

Exception

JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

Example of Exception implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
```

```
%>
```

application

In JSP, application is an implicit object of type ServletContext. this application object in the Servlet programming is ServletContext.

For all jsp in a web application, there must be a single application object with application object we can share the data from one JSP to any other JSP in the web application.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

Example of Application implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

web.xml

```
<web-app>

<servlet>
<servlet-name>One</servlet-name>
<jsp-file>/welcome.jsp</servlet-class>
</servlet>
```

```
<servlet-mapping>
<servlet-name>One</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

</web-app>
```

pageContext

In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes.

- page
- request
- session
- application

In JSP, page scope is the default scope.

Example of pageContext implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
<%
```

```
String name=request.getParameter("uname");
out.print("Welcome "+name);

pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

<a href="One.jsp">One jsp page</a>

%>
```

One.jsp

```
String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);
```

JSP Action Element

Action elements are used to performs specific operation using JSP container, like setting values into java class, getting values from java class. The JSP action elements classified into two types are;

- JSP Standard Action Element
- JSP Custom Action Element

Standard Action Element

Standard Action are given in JSP to separate the presentation logic and the business logic of the JSP but partial.
The name is given as a standard Action, because each action as a pre-defined meaning and as a programmer it is not possible to change the meaning of the tag.
Each Standard Action follows xml syntax, we do not have any equivalent html syntax.
The standard action element followed by "JSP" prifix.

Syntax of Standard Action Element

```
<jsp: standard action name>
```

Standard Action are given by JSP are;

- <jsp: include>
- <jsp: forward>
- <jsp: param>
- <jsp: params>
- <jsp: plugin>
- <jsp: useBean>
- <jsp: setProperty>
- <jsp: getProperty>
- <jsp: fallback>

Session Tracking

Session is the conversion of user within span of time. In general meaning particular interval of time.

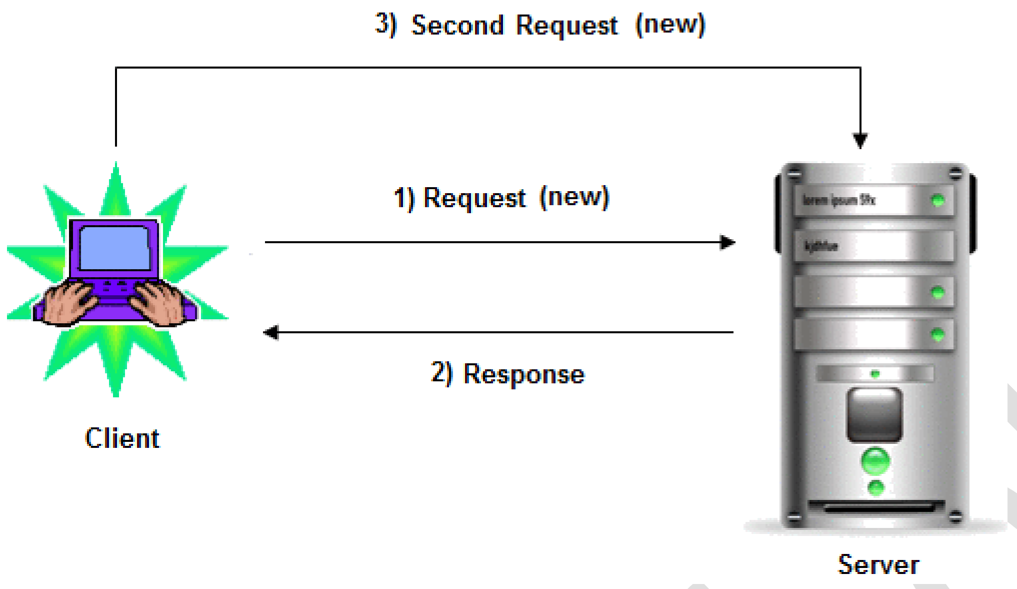
Tracking is the recording of the thing under session.

Session Tracking is remembering and recording of client conversion in span of time. It is also called as session management.

If web application is capable of remembering and recording of client conversion in span of time then that web application is called as **stateful web application**.

Why need Session Tracking ?

- Http protocol is stateless, to make stateful between client and server we need Session Tracking.
- Session Tracking is useful for online shopping, mailing application, E-Commerce application to track the conversion.
- Http protocol is stateless, that means each request is considered as the new request. You can see in below image.



Why use Session Tracking ?

To recognize the user It is used to recognize the particular user.

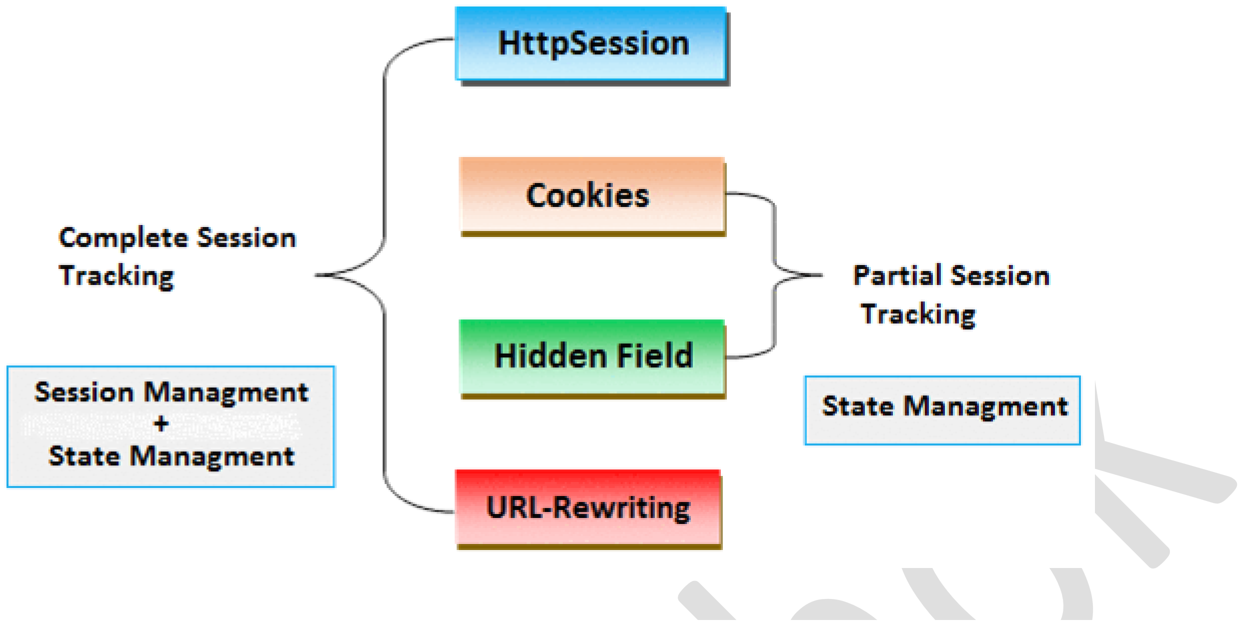
Why Http is design as stateless protocol ?

If Http is stateful protocol for multiple requests given by client to web application single connection will be used between browser and web server across the multiple requests. This may make clients to engage connection with web server for long time event though the connection are ideal. Due to this the web server reach to maximum connections even though most of its connection are idle. To overcome this problem Http is given as stateless.

Session Tracking Techniques

Servlet technology allows four technique to track conversion, they are;

- Cookies
- URL Rewriting
- Hidden Form Field
- HttpSession



Cookies

Cookies are text files stored on the client computer and they are kept for various information like name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookies are created using Cookie class present in Servlet API. Cookies are added to response object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used access the cookies that are added to response object.

In Http Session technique, container internally generates a cookies for transferring the session ID between server and client. Apart from container generated cookie a servlet programmer can also generate cookies for storing the data for a client.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser (chrome, firefox) at client side. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

When to use cookies ?

When session ID is not required and when less number of input values are submitted by client in that case in place of using HttpSession Technique you can use cookies Technique to reduce the burden on server.

Points to Remember

- Cookies is pressistance resource which is stores at client location.
- We can store 3000 cookies in cookies file at a time.
- The cookies are introduced by net scape communication.

- Cookies files exist up to 3 year.
- Size of cookies is 4 kb.

Type of Coockies

There are two types of cookies, those are given below;

- In-memory cookies or pre session cookies
- Persistent cookies

In-memory cookies: By default cookie is in-memory coockie, This type of cookie is lives until that browser is destroy(close). It is valid for **single session** only. It is removed each time when user closes the browser.

Persistent cookies: Presestent cookie lives on a browser until its expiration time is reached it means , eventhough you close or reopen the browser but still the cookie exists on the browser. It is valid for **multiple session**. It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a some constructor and methods for cookies.

Cookies are text files stored on the client computer and they are kept for various information like name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookies are created using Cookie class present in Servlet API. Cookies are added to response object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used access the cookies that are added to response object.

In Http Session technique, container internally generates a cookies for transferring the session ID between server and client. Apart from container generated cookie a servlet programmer can also generate cookies for storing the data for a client.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser (chrome, firefox) at client side. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

When use cookies ?

When session ID is not required and when less number of input values are submitted by client in that case in place of using HttpSession Technique you can use cookies Technique to reduce the burden on server.

Points to Remember

- Cookies is pressistance resource which is stores at client location.
- We can store 3000 cookies in cookies file at a time.
- The cookies are introduced by net scape communication.
- Cookies files exist up to 3 year.
- Size of cookies is 4 kb.

To create cookies you need to use Cookie class of javax.servlet.http package.

```
Cookie c=new Cookie(name, value);
// here name and value are string type
```

Add Cookies

To add a cookie to the response object, we use addCookie() mehtod.

```
Cookie c=new Cookie();     //creating cookie object
response.addCookie(c1); //adding cookie in the response
```

Read Cookies for browser

To read Cookies from browser to a servlet, we need to call getCookies methods given by request object and it returns an array type of cookie class.

```
response.addCookie(c1);
Cookie c[]=request.getCookie();
```

Advantage of Cookie

- Simplest technique of maintaining the state.
- Cookie are maintained at client side so they do not give any burden to server.
- All server side technology and all web server, application servers support cookies.
- Presistent cookies can remember client data during session and after session with expiry time.

Limitation of Coockie

- It will not work if cookie is disabled from the browser.
- Cookies are text files, It does not provides security. Any one can change this file.
- With coockies need client support that means if client disable the coockies then it does not store the client location.
- Cookies can not store java objects as values, they only store text or string.

Example of session tracking by using Cookies

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
 public void doPost(HttpServletRequest request, HttpServletResponse response){
   try{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   String n=request.getParameter("userName");
   out.print("Welcome "+n);
    Cookie ck=new Cookie("uname",n);//creating cookie object
   response.addCookie(ck);//adding cookie in the response
   //creating submit button
   out.print("<form action='servlet2'>");
   out.print("<input type='submit' value='continue'>");
   out.print("</form>");
   out.close();
       }catch(Exception e){System.out.println(e);}
 }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());
    out.close();
   }catch(Exception e){System.out.println(e);}
    }
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
```

```
</servlet-mapping>
</web-app>
```

Hidden Form Field

Tracking client conversion using Html hidden variables in secure manner is known as hidden form field.

How to use Hidden Form Field ?

In Hidden Form Field we are use html tag is <input type="hidden"> and with this we assign session ID value.

```
<input type="hidden" name="uname" value="porter">
```

Hidden Form Field Advantage

- Basic knowledge of html is enough to work with this technique.
- It will always work whether cookie is disabled or not.
- Hidden boxes resides in web pages of browser window so they do not provide burden to the server.
- This technique can be used along with all kind of web server or application server.

Hidden Form Field Dis-Advantage

- More complex than URL Rewriting.
- It is maintained at server side.
- Extra form submission is required on each pages.
- Hidden form field can not store java object as values. They only store text value
- It Also increase network traffic because hidden boxes data travels over the network along with request and response.
- Hidden boxes does not provides data security because their data can be view through view source option.

Example of session tracking by using Hidden Form Field

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
```

```
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{
     response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    //creating form that have invisible textfield
    out.print("<form action='servlet2'>");
    out.print("<input type='hidden' name='uname' value='"+n+"'>");
    out.print("<input type='submit' value='continue'>");
    out.print("</form>");
    out.close();
      }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    //Getting the value from the hidden field
    String n=request.getParameter("uname");
    out.print("Hello "+n);
```

```
        out.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

URL Rewriting

URL Rewriting **track the conversion in server based on unique session ID value.**

When to use URL Rewriting ?

If the client has disabled cookie in the browser then coockie are not work for session management. In that case you can use URL rewriting technique for session managment. URL rewriting will always work.

Advantage of URL Rewriting

- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages

Dis-advantage of URL Rewriting

- Generate more network traffic.
- It will work only with links.
- It can send Only textual information.
- Less secqure because query string in session id displace on address bar.

Example of session tracking by using URL Rewriting

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    HttpSession session=request.getSession();
    session.setAttribute("uname",n);
    out.print("<a href='servlet2'>visit</a>");
    out.close();
   }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    //getting value from the query string
    String n=request.getParameter("uname");
    out.print("Hello "+n);
     out.close();
     }catch(Exception e){System.out.println(e);}
  }
}

<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

HttpSession

HttpSession is another kind of session management technique, In this technique create a session object at server side for each client. Session is available until the session time out, until the client log out. The default session time is 30 minutes and can configure explicit session time in web.xml file. Configure session time in web.xml

```
<web-app>
<session-config>
<session-timeout>40</session-timeout>
</session-config>
</web-app>
```

HttpSession Api

Http session is an interface define in java.http package. Getting session object

```
HttpSession hs=req.getSession();  // create new session object
```

Example of session tracking by using httpsession

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/> <br/>
<input type="submit" value="continue"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response){
       try{
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       String n=request.getParameter("userName");
       out.print("Welcome "+n);
       HttpSession session=request.getSession();
       session.setAttribute("uname",n);
       out.print("<a href='servlet2'>visit</a>");
       out.close();
         }catch(Exception e){System.out.println(e);}
    }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
       try{
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       HttpSession session=request.getSession(false);
       String n=(String)session.getAttribute("uname");
       out.print("Hello "+n);
         out.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
```

```
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
</web-app>
```

Expression Language (EL Expressions)

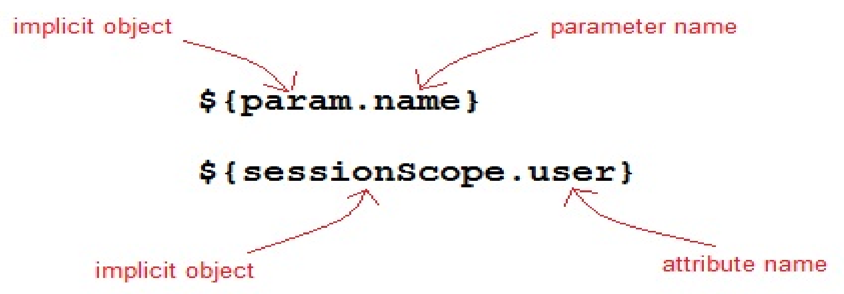
Most of the times we use JSP for view purposes and all the business logic is present in servlet code or model classes. When we receive client request in servlet, we process it and then add attributes in request/session/context scope to be retrieved in JSP code. We also use request params, headers, cookies and init params in JSP to create response views.

We saw in earlier posts, how we can use **scriptlets** and **JSP expressions** to retrieve attributes and parameters in JSP with java code and use it for view purpose. But for web designers, java code is hard to understand and that's why JSP Specs 2.0 introduced **Expression Language** (EL) through which we can get attributes and parameters easily using HTML like tags.

The purpose of EL is to produce scriptless JSP pages. The syntax of EL in a JSP is as follows:

```
${expr}
```

Here expr is a valid EL expression. An expression can be mixed with static text/values and can also be combined with other expressions to form larger expression.



How EL expression is used?

EL expression can be used in two ways in a JSP page

As attribute values in standard and custom tags. Example:

<jsp:include page="${location}">

Where location variable is separately defines in the jsp page.

Expressions can also be used in jsp:setProperty to set a properties value, using other bean properties like : If we have a bean named Square with properties length, breadth and area.

<jsp:setProperty name="square" property="area" value="${square.length*square.breadth}" />

To output in HTML tag : <h1>Welcome ${name}</h1>

To deactivate the evaluation of EL expressions, we specify the isELIgnored attribute of the page directive as below:

<%@ page isELIgnored ="true|false" %>

Implicit Objects in Expression Language

JSP Expression Language provides many implicit objects that we can use to get attributes from different scopes and parameter values. The list is given below.

Implicit Object	Description
pageContext	It represents the PageContext object.
pageScope	It is used to access the value of any variable which is set in the **Page** scope
requestScope	It is used to access the value of any variable which is set in the **Request** scope.
sessionScope	It is used to access the value of any variable which is set in the **Session** scope
applicationScope	It is used to access the value of any variable which is set in the **Application** scope
param	Map a request parameter name to a single value

paramValues	Map a request parameter name to corresponding array of string values.
header	Map containing header names and single string values.
headerValues	Map containing header names to corresponding array of string values.
cookie	Map containing cookie names and single string values.

```
-----------------------------------------------------index.jsp-----------------------------------------------------------

          <form method="POST" action="welcome.jsp">
             Name <input type="text" name="user" >
             <input type="submit" value="Submit">
          </form>
-----------------------------------------------------welcome.jsp-------------------------------------------------------------------

          <html>
             <head>
                <title>Welcome Page</title>
             </head>

            <body>
                <h1>Welcome ${param.name}</h1>
            </body>
          </html>
-----------------------------------------------------------0-------------------------------------------------------------------------------------
```

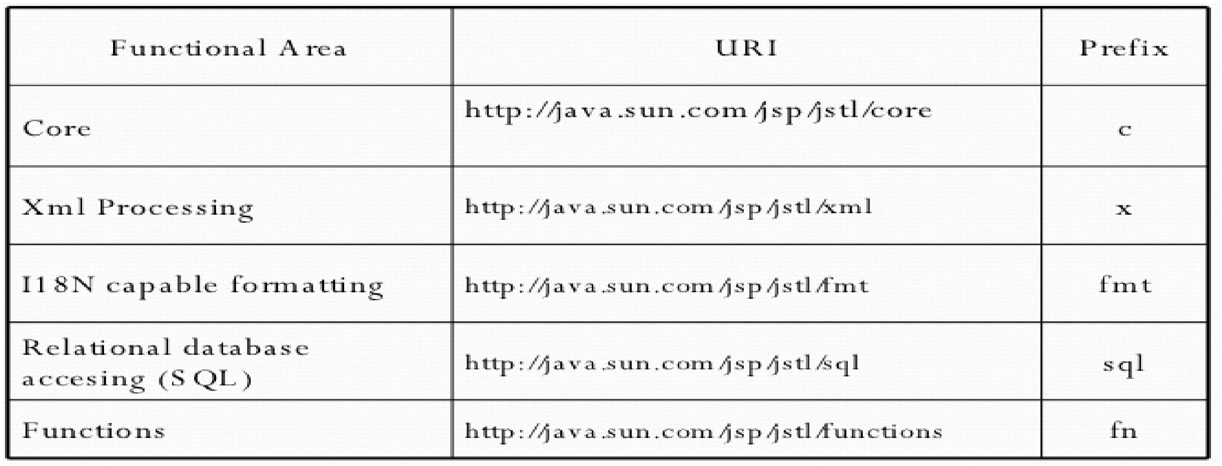
JSP – Standard Tag Library (JSTL)

JSTL stands for The JSP standard Tag Library is used to simplify the JSP development to represent a set of tags. It gives support to iteration and conditional tasks, underpins the tags to control the xml files, it underpins skeleton to the merged existed custom tags. The following are the advantages of JSTL.

- It have many tags for fast development which simplifies the JSP.
- It was used for reusability of the code in various pages.
- For using JSTL, scriptlet tag can be avoided.

The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

❖ Core Tags	→ used for import,if, foreach loops
❖ Formatting tags	→ used for formatting text, Date,number,URLencoding
❖ SQL tags	→ used for SQL operations like INSERT,SELECT., etc
❖ XML tags	→ provides support for XML processing
❖ JSTL Functions	→ provides support for string manipulation.



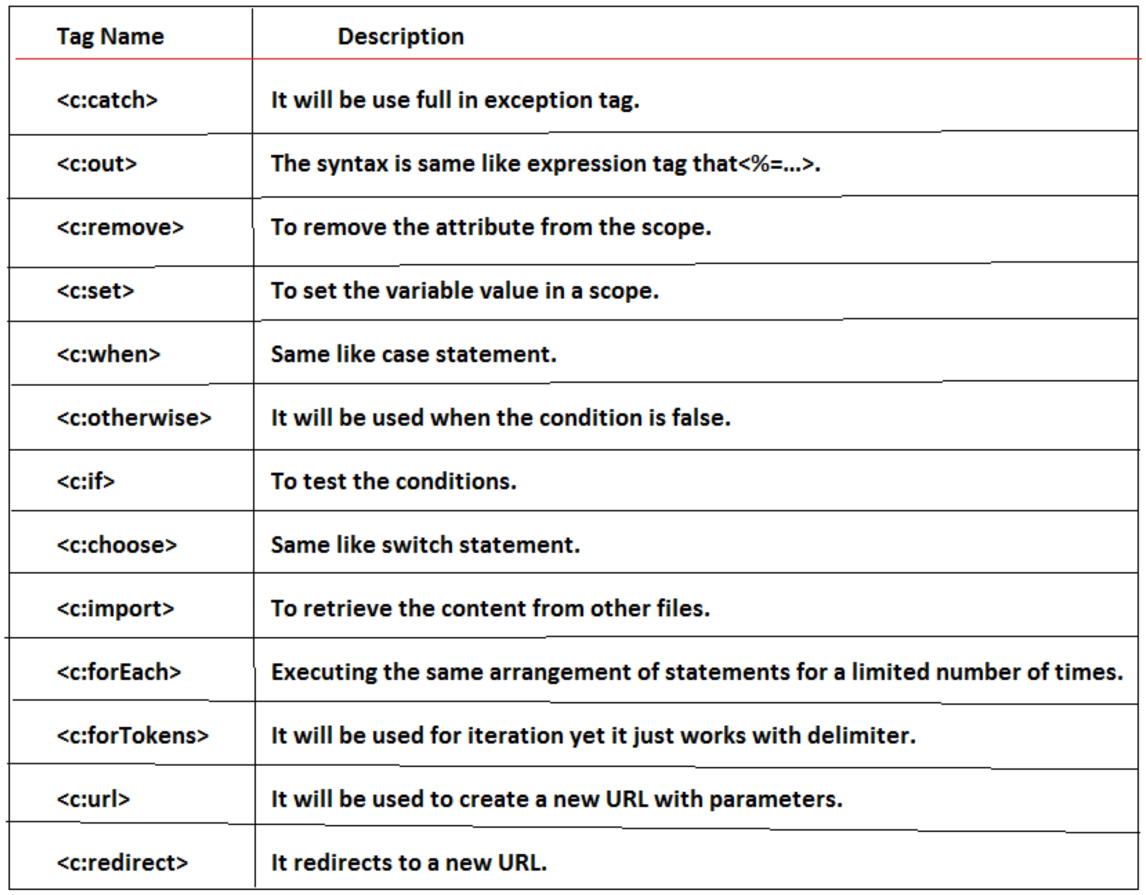
Functional Area	URI	Prefix
Core	http://java.sun.com/jsp/jstl/core	c
Xml Processing	http://java.sun.com/jsp/jstl/xml	x
I18N capable formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Relational database accesing (SQL)	http://java.sun.com/jsp/jstl/sql	sql
Functions	http://java.sun.com/jsp/jstl/functions	fn

Core tags

Core tags provide variable support, flow control and URL administration and so on

The following is the syntax to represent core tags.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```



Tag Name	Description
<c:catch>	It will be use full in exception tag.
<c:out>	The syntax is same like expression tag that<%=…>.
<c:remove>	To remove the attribute from the scope.
<c:set>	To set the variable value in a scope.
<c:when>	Same like case statement.
<c:otherwise>	It will be used when the condition is false.
<c:if>	To test the conditions.
<c:choose>	Same like switch statement.
<c:import>	To retrieve the content from other files.
<c:forEach>	Executing the same arrangement of statements for a limited number of times.
<c:forTokens>	It will be used for iteration yet it just works with delimiter.
<c:url>	It will be used to create a new URL with parameters.
<c:redirect>	It redirects to a new URL.

Function Tags

Function tags gives support for sting manipulation.

URL tag is http://java.sun.com/jsp/jstl/functions and where fn is the prefix.

Tag Name	Description
fn:contains()	To inspect whether the given string is available in the input as sub-string.
fn:containsIgnoreCase()	It does a case uncaring verify whether the permitted string is a sub-string of information.
fn:indexOf()	It is utilized for discovering the begin position of a string in the endowed string. Returns -1 if string is not identified in the information.
fn:split():	To split the string as substring.
fn:join()	It is used to Connect entirely components of an array into a string.
fn:escapeXML()	It is utilized to dodge the characters that could be decoded as XML markup.
fn:length()	To find the length of string and strives the components available in the collection.
fn:startsWith()	It is used to check if an information string initiates with the predetermined prefix.
fn:endsWith()	To check the suffix of the string.
fn:trim()	To vanish the gap of the string at staring and at ending.
fn:substring()	To get the substing from the main string.
fn:substringAfter()	Gives back a subset of a string taking after a particular substring.
fn:substringAfter()	Strive for string in the information and supplant it with the supplied string.

JSTL SQL Tags

Provides SQL support and the URL for SQL tag is http://java.sun.com/jsp/jstl/sql **and where prefix is** sql**.**

Tag Name	Description
<sql:setDataSource>	Creates simple data source.
<sql:query>	Accomplishes the SQL query characterized in its body.
<sql:param>	Providing the parameters to sql query.
<sql:update>	Accomplishes the SQL update characterized in its body.
<sql:dateparam>	Provide the parameters to specified java.util.Date.
<sql:transaction>	Gives settled database activity components with a common Connection, set up to execute all transactions as one exchange.

JSTL Internationalization Tags / Formatting tags

The tag gives backing to message formating, number and date designing and so on. What's more, the url's tag is http://java.sun.com/jsp/jstl/fmt and where fmt is the prefix.

Tag Name	Description
<fmt:requestEncoding>	To encode the characters of a solicitation.
<fmt:message>	To show formatted messages.
<fmt:parseDate>	To parse the date and time of a string.
<fmt:formatDate>	Positions a date and/or time utilizing the conveyed styles.
<fmt:setBundle>	Loads a benefit pack and stores in the variable.
<fmt:setLocale>	Stores the locale in the variable.
<fmt:bundle>	Loads an asset pack to be utilized by its label body.
<fmt:timeZone>	Used to determine timezone for at whatever time formatting.
<fmt:setTimeZone>	Stores the time zone in a variable.
<fmt:parseNumber>	Analyze the string illustration of a number.
<fmt:formatNumber>	To depict statistical value with particular exactness.

Custom Tags in JSP

When EL and Standard Action elements aren't enough to remove scriptlet code from your JSP Page, you can use Custom Tags. Custom tags are nothing but user-defined tags.

Custom tags are an excellent way to abstract the complexity of business logic from the presentation of Web pages in a way that is easy for the Web author to use and control. It also allows for reusability as custom tags can be used again and again.

1) Tag handler class: In this class we specify what our custom tag will do when it is used in a JSP page.
2) TLD file: Tag descriptor file where we will specify our tag name, tag handler class and tag attributes.
3) JSP page: A JSP page where we will be using our custom tag.

Tag Handler Class

We can create a Tag Handler class in two different ways:

By implementing one of three interfaces : SimpleTag, Tag or BodyTag, which define methods that are invoked during the life cycle of the tag.

By extending an abstract base class that implements the SimpleTag, Tag, or BodyTag interfaces. The SimpleTagSupport, TagSupport, and BodyTagSupport classes implement the SimpleTag, Tag andBodyTag interfaces . Extending these classes relieves the tag handler class from having to implement all methods in the interfaces and also provides other convenient functionality.

Tag Library Descriptor

A Tag Library Descriptor is an XML document that contains information about a library as a whole and about each tag contained in the library. TLDs are used by the web container to validate the tags and also by JSP page development tools.

Tag library descriptor file must have the extension .tld and must be packaged in the /WEB-INF/ directory or subdirectory of the WAR file or in the /META-INF/ directory or subdirectory of a tag library packaged in a JAR.

Example of Custom Tag

In our example, we will be creating a Tag Handler class that extends the TagSupport class. When we extend this class, we have to override the method doStartTag(). There are two other methods of this class namely doEndTag() and release(), that we can decide to override or not depending on our requirement.

---CountMatches.java---

```java
package com.ashoksoft.apps;

import java.io.IOException;
import javax.servlet.jsp.*;
import org.apache.commons.lang.StringUtils;

public class CountMatches extends TagSupport {
	private String inputstring;
	private String lookupstring;

	public String getInputstring() {
		return inputstring;
	}

	public void setInputstring(String inputstring) {
		this.inputstring = inputstring;
	}
	public String getLookupstring() {
		return lookupstring;
	}

	public void setLookupstring(String lookupstring) {
		this.lookupstring = lookupstring;
	}

	@Override
	public int doStartTag() throws JspException {
		try {
			JspWriter out = pageContext.getOut();
			out.println(StringUtils.countMatches(inputstring, lookupstring));
		} catch (IOException e) {
			e.printStackTrace();
		}
		return SKIP_BODY;
	}
}
-----------------------------------------0-----------------------------------------
```

In the above code, we have an implementation of the doStartTag() method which is must if we are extending TagSupport class. We have declared two variables inputstring and lookupstring. These variables represents the attributes of the custom tag. We must provide getter and setter for these variables in order to set the values into these variables that will be provided at the time of using this custom tag. We can also specify whether these attributes are required or not.

```
------------------------------------MyTags.tld------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>cntmtchs</shortname>
  <info>Sample taglib for Substr operation</info>
  <uri>http://studytonight.com/jsp/taglib/countmatches</uri>

  <tag>
    <name>countmatches</name>
    <tagclass>com.studytonight.taghandler.CountMatches</tagclass>
    <info>String Utility</info>
    <attribute>
       <name>inputstring</name>
       <required>true</required>
    </attribute>
    <attribute>
       <name>lookupstring</name>
       <required>true</required>
    </attribute>
  </tag>
</taglib>
```

The taglib element specifies the schema, required JSP version and the tags within this tag library. Each tag element within the TLD represents an individual custom tag that exist in the library. Each of these tag should have a tag handler class associated with them.

The uri element represents a Uniform Resource Identifier that uniquely identifies the tag library.

The two attribute elements within the tag element represents that the tag has two attributes and the true value provided to the required element represents that both of these attributes are required for the tag to function properly.

```
-----------------------------------------index.jsp---------------------------------------
<%@taglib prefix="mytag" uri="/WEB-INF/CountMatchesDescriptor.tld"%>
<body>
	<h3>Custom Tag Examole</h3>
	<mytag:countmatches inputstring="ashoksoft" lookupstring="o" />
</body>

</html>
-----------------------------------------0-----------------------------------------------
```

If this tag works fine it should print a value 3 in the browser as there 't' occurs 2 times in the word “ashoksoft”..

Connection Pooling in Web Applications

The Java 2 Enterprise Edition (J2EE) specification provides a distributed services-based architecture for implementing highly scalable, reliable applications. In general, a J2EE application architecture maps to the Model-View-Controller (MVC) framework.

A typical business application use case would be realized by components in all the three layers on the server side. Given the large number of user interactions (millions for customer-facing applications), the finite server-side resources need to be optimally shared. Such resources may include databases, message queues, directories, enterprise systems (SAP, CICS), and so forth, each of which is accessed by an application using a connection object that represents the resource entry point. Managing access to those shared resources is essential for meeting the high-performance requirements for J2EE applications.

The process of creating a connection, always an expensive, time-consuming operation, is multiplied in these environments where a large number of users are accessing the database. Creating connections over and over in these environments is too expensive and reduce performance of the application.

Connection pooling is a technique that was pioneered by database vendors to allow multiple clients to share a cached set of connection objects that provide access to a database resource. Connection pooling has become the standard for middleware database drivers.

What is Connection Pooling?

A cache of database connections maintained in the pool memory, so that the connections can be reused when user request for the connection

Particularly for server-side web applications, a connection pool is the standard way to maintain a “pool” of active database connections in memory which are reused across requests.

JNDI

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) for accessing different kinds of naming and directory services. JNDI is not specific to a particular naming or directory service, it can be used to access many different kinds of systems including file systems; distributed objects systems like CORBA, Java RMI, and EJB; and directory services like LDAP, Novell NetWare, and NIS+.

JNDI is similar to JDBC in that they are both Object-Oriented Java APIs that provide a common abstraction for accessing services from different vendors. While JDBC can be used to access a variety of relational databases, JNDI can be used to access a variety of of naming and directory services.

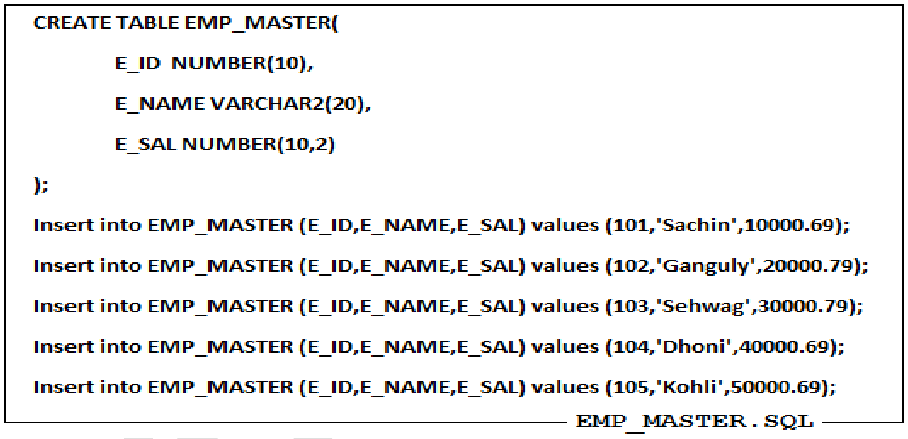
The benefits of using a JNDI DataSource

There are major advantages to connecting to a data source using a DataSource object registered with a JNDI naming service rather than using the DriverManager facility.

- ➢ The first is that it makes code more portable. (With the DriverManager, the name of a JDBC driver class, which usually identifies a particular driver vendor, is included in application code. This makes the application specific to that vendor's driver product and thus non-portable)
- ➢ It makes code much easier to maintain. If any of the necessary information about the data source changes, only the relevant DataSource properties need to be modified
- ➢ Multiple wars can use the same connection pool which might be better use of resources
- ➢ It's a fine way to externalize configuration.

Configuring JNDI DataSource for Database Connection Pooling in Tomcat

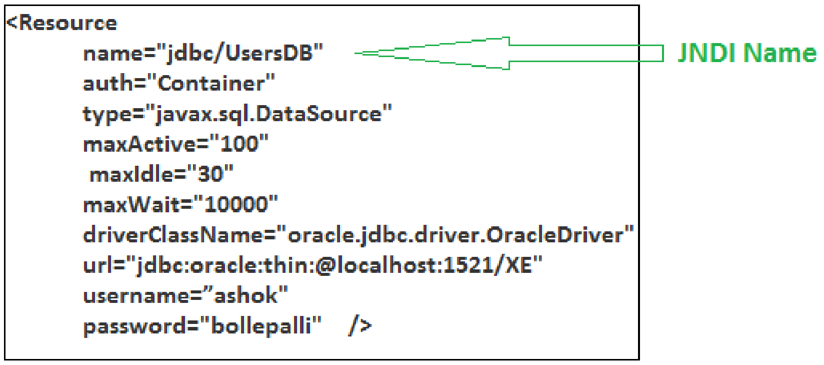
Step 1: Create a table in Oracle Database and insert the few records using below SQL Queries



```
CREATE TABLE EMP_MASTER(
        E_ID  NUMBER(10),
        E_NAME VARCHAR2(20),
        E_SAL NUMBER(10,2)
);
Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (101,'Sachin',10000.69);
Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (102,'Ganguly',20000.79);
Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (103,'Sehwag',30000.79);
Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (104,'Dhoni',40000.69);
Insert into EMP_MASTER (E_ID,E_NAME,E_SAL) values (105,'Kohli',50000.69);
```

Step 2: Configuring Context Details

To declare a JNDI DataSource for the Oracle database, create a **Resource** XML element with the following content



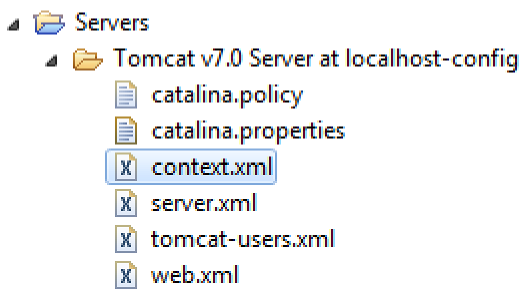
```
<Resource
        name="jdbc/UsersDB"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100"
         maxIdle="30"
        maxWait="10000"
        driverClassName="oracle.jdbc.driver.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521/XE"
        username="ashok"
        password="bollepalli"    />
```

Note: Change Database Username and password according to your DB details

Add this element inside the root element <Context> in a **context.xml** file. There are two places where the context.xml file can reside (create one if not exist):

1. Inside **/META-INF** directory of a web application: the JNDI DataSource is only available to the application itself, thus it cannot be shared among other ones. In addition, this makes the configuration dependent on the application.
2. Inside **$CATALINA_BASE/conf** directory (server conf folder): this is the preferred place because the JNDI DataSource will be available to all web applications and it's independent of any applications.

If you are using Tomcat inside Eclipse IDE, you need to modify the context.xml file under the Servers project. That is because Eclipse made a copy of Tomcat configuration:



The following table describes the attributes specified in the above configuration:

Attribute name	Description
name	Name of the resource.
auth	Specify authentication mechanism for the application code, can be Application or Container.
type	The fully qualified Java class name expected by the web application when it performs a lookup for this resource.
maxActive	Maximum number of database connections in pool. Set to -1 for no limit.
maxIdle	Maximum number of idle database connections to retain in pool. Set to -1 for no limit.
maxWait	Maximum time to wait for a database connection to become available in ms, in this example 10 seconds. An Exception is thrown if this timeout is exceeded. Set to -1 to wait indefinitely.
driverClassName	The fully qualified Java class name of the database driver
url	The JDBC connection URL.
username	Database user name.
password	Database user password.

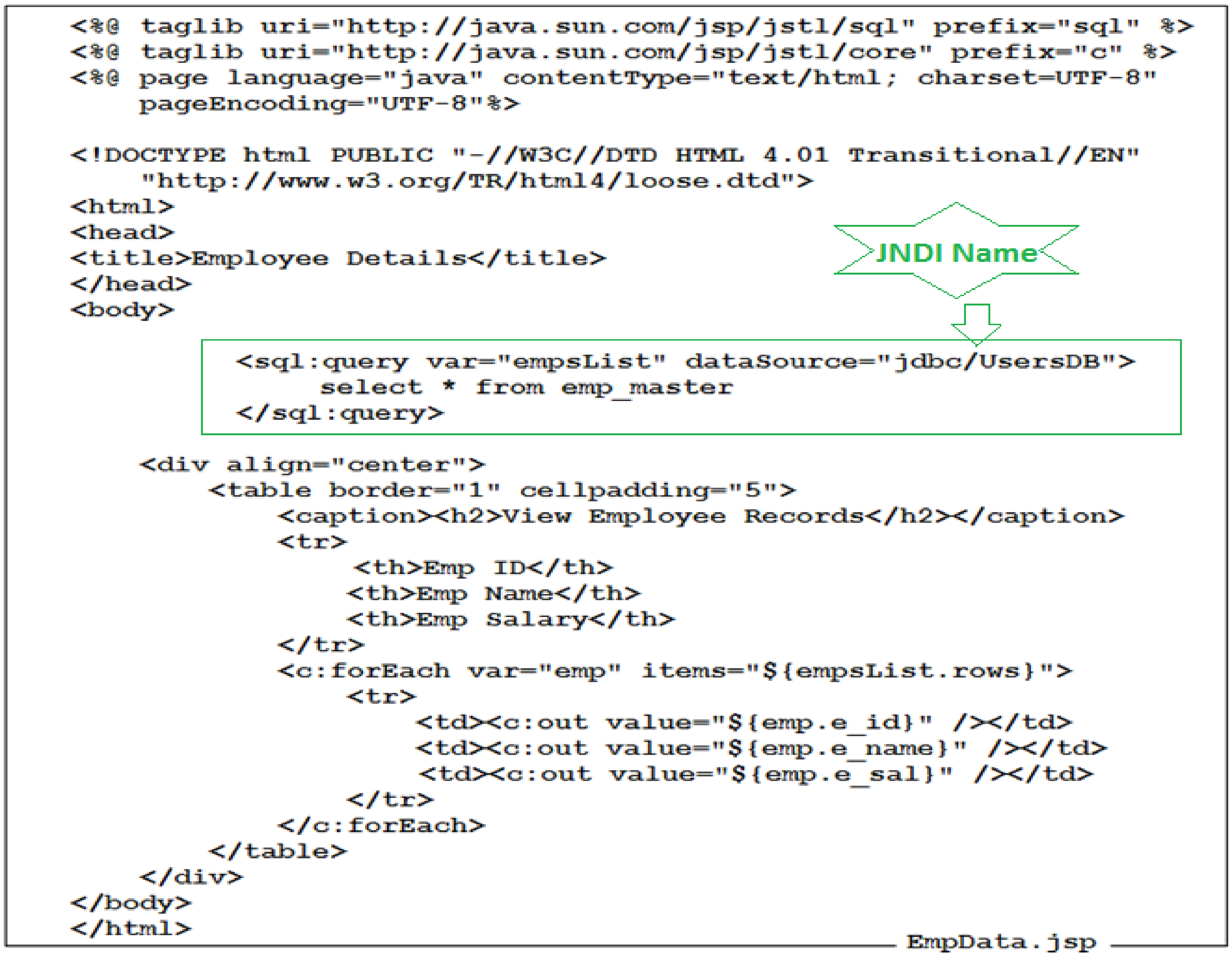
Step 3: Configuring resource ref details in project web.xml file

Add the following declaration in project web.xml file

```
<resource-ref>
                <description>DB Connection</description>
                <res-ref-name>jdbc/UsersDB</res-ref-name>
                 <res-type>javax.sql.DataSource</res-type>
                 <res-auth>Container</res-auth>
</resource-ref>
```

This is necessary in order to make the JNDI DataSource available to the application under the specified namespace jdbc/UsersDB

Step 4: Create JSP Page in web application to retrieve Employees data from Database with JNDI DataSource



```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Details</title>
</head>
<body>

        <sql:query var="empsList" dataSource="jdbc/UsersDB">
            select * from emp_master
        </sql:query>

    <div align="center">
        <table border="1" cellpadding="5">
            <caption><h2>View Employee Records</h2></caption>
            <tr>
                <th>Emp ID</th>
                <th>Emp Name</th>
                <th>Emp Salary</th>
            </tr>
            <c:forEach var="emp" items="${empsList.rows}">
                <tr>
                    <td><c:out value="${emp.e_id}" /></td>
                    <td><c:out value="${emp.e_name}" /></td>
                    <td><c:out value="${emp.e_sal}" /></td>
                </tr>
            </c:forEach>
        </table>
    </div>
</body>
</html>
```

Note: Here, we use the JSTL's SQL tag **query** to make a SELECT query to the database. Note that the **DataSource** attribute refers to the JNDI resource name declared in the web.xml file.

Getting JNDI DataSource in Java Program

We can look up the configured JNDI DataSource using Java code as follows:

```
Context initContext = new InitialContext();
Context envContext = (Context) initContext.lookup("java:comp/env");
DataSource ds = (DataSource) envContext.lookup("jdbc/UsersDB");
Connection conn = ds.getConnection();
```

After obtaining the connection, we can write JDBC code using this conn object

Getting JNDI DataSource using annotations

Alternatively, we can use the **@Resource** annotation (**javax.annotation.Resource**) instead of the lookup code above. For example, declare a field called **dataSource** in the servlet like this:

```
@Resource(name = "jdbc/UsersDB")
private DataSource dataSource;
```

Tomcat will look up the specified resource name and inject an actual implementation when it discovers this annotation. Therefore, the servlet code looks like this:

```
@WebServlet("/retrieveEmpData")
public class EmpListServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Resource(name = "jdbc/UsersDB")
    private DataSource dataSource;          //Getting JNDI DataSource

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
             PrintWriter writer = response.getWriter();
        try {
            Connection conn = dataSource.getConnection();  //getting Connection

            Statement statement = conn.createStatement();
            String sql = "select * from emp_master";
            ResultSet rs = statement.executeQuery(sql);

            while (rs.next()) {
                writer.write(rs.getInt("e_id");
                writer.write(rs.getInt("e_name");
                writer.write(rs.getInt("e_sal");
                writer.write(rs.getInt("<br/>");
            }
        } catch (SQLException ex) {
            System.err.println(ex);
        }
    }
}
```
EmpListServlet.java

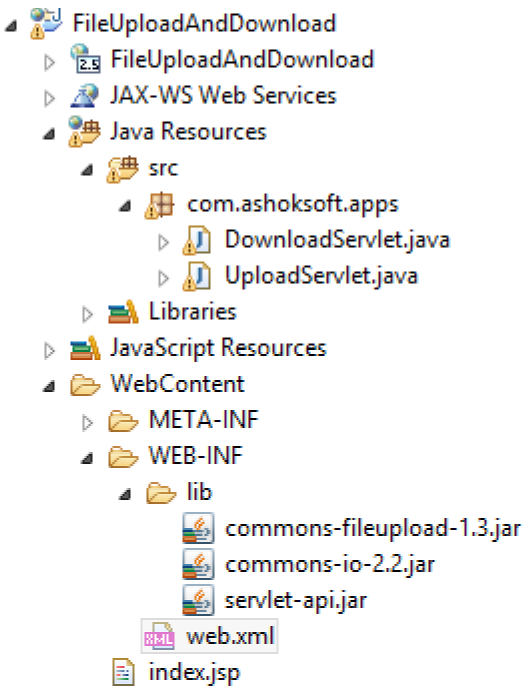
File Uploading and Downloading Using Servlets & JSP

File Upload and Download is always a handy utility to know. There will be some need to upload a file to an FTP server, Like if you generate a report or store some data in .xls file, then it needs to be uploaded to a FTP server for further use. like wise we need to download some data (data stored in .xls files)for manuplation from the server in our projects. Here we have the code to do this for us. The FileUploadDownload utility.

If we want user to choose files from file system and upload to server then following are the important points to be noted down:

- You need to use <input type="file"/>
- Next step is that form method should be POST with enctype as multipart/form-data, which makes file data available in parts inside request body.
- The form action attribute should be set to a servlet file which would handle file uploading at backend server.

Project Structure



```
FileUploadAndDownload
  FileUploadAndDownload
  JAX-WS Web Services
  Java Resources
    src
      com.ashoksoft.apps
        DownloadServlet.java
        UploadServlet.java
    Libraries
  JavaScript Resources
  WebContent
    META-INF
    WEB-INF
      lib
        commons-fileupload-1.3.jar
        commons-io-2.2.jar
        servlet-api.jar
      web.xml
    index.jsp
```

```
=====================================index.jsp=============================================
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
      pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
</head>
<body>
      <form action="UploadServlet" enctype="multipart/form-data"
            method="post">
            <table>
                  <tr>
                        <td>Enter Name :</td>
                        <td><input type="text" name="uname" /></td>
                  </tr>
                  <tr>
                        <td>Select Resume :</td>
                        <td><input type="file" name="file1" /></td>
                  </tr>
                  <tr>
                        <td colspan="2"><input type="submit" value="Upload"
/></td>
                  </tr>
            </table>
      </form>
</body>
</html>
==========================UploadServlet.java==========================
package com.ashoksoft.apps;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

public class UploadServlet extends HttpServlet {

     protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException,
IOException {
           response.setContentType("text/html");
           PrintWriter out = response.getWriter();

           ServletContext context = request.getServletContext();
           String filesPath = context.getInitParameter("filesPath");

           DiskFileItemFactory factory = new DiskFileItemFactory();
           ServletFileUpload upload = new ServletFileUpload(factory);

           try {
                List<FileItem> list = upload.parseRequest(request);
                Iterator<FileItem> itr = list.iterator();
                String uname = "";
                while (itr.hasNext()) {
                     FileItem fi = itr.next();

                     if (fi.isFormField()) {
                          String fieldName = fi.getFieldName();
                          if (fieldName.equals("uname")) {
                               uname = fi.getString();
                          }
                     } else {
                          String loc = fi.getName();
                          String fileName =
loc.substring(loc.lastIndexOf("\\") + 1);
                          File f = new File(filesPath + File.separator +
fileName);
                          fi.write(f);
                          out.print("File uploaded successfully");
                          out.write("<a href=\"DownLoadServlet?fileName="
                                    + fi.getName() + "\">Download " +
f.getName()
                                    + "</a>");
```

```
                        }
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
        }
}
```

-- **DownloadServlet.java**---

```
package com.ashoksoft.apps;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DownloadServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {

                response.setContentType("APPLICATION/OCTET-STREAM");
                String fileLoc = request.getParameter("filename");

                File f = new File(fileLoc);

                response.setHeader("Content-Dispostion",
                            "attachment;filename=\"" + f.getName() + "\"");

                FileInputStream fis = new FileInputStream(f);

                PrintWriter out = response.getWriter();

                int i = fis.read();
                while (i != -1) {
                        out.write(i);
                        i = fis.read();
                }
                fis.close();
                out.close();
        }
}
```

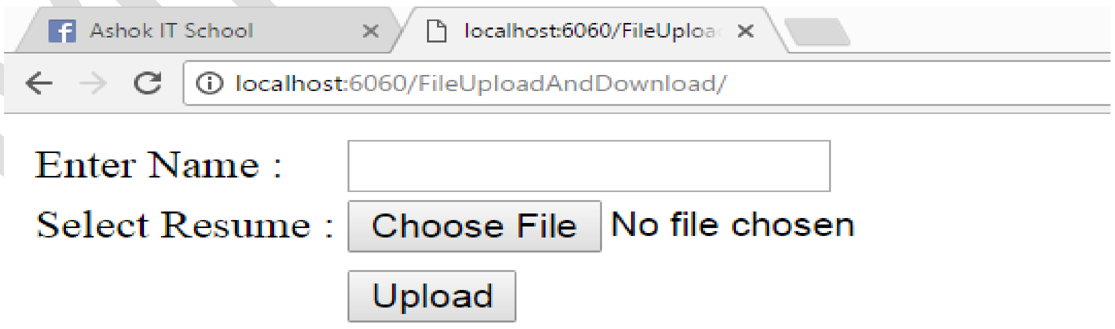
======================================**web.xml**===

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.5"
        xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
	<display-name>FileUploadAndDownload</display-name>
	<servlet>
		<servlet-name>UploadServlet</servlet-name>
		<servlet-class>com.ashoksoft.apps.UploadServlet</servlet-class>
	</servlet>
	<servlet>
		<display-name>DownloadServlet</display-name>
		<servlet-name>DownloadServlet</servlet-name>
		<servlet-class>com.ashoksoft.apps.DownloadServlet</servlet-
class>
	</servlet>
	<servlet-mapping>
		<servlet-name>UploadServlet</servlet-name>
		<url-pattern>/UploadServlet</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>DownloadServlet</servlet-name>
		<url-pattern>/DownloadServlet</url-pattern>
	</servlet-mapping>
	<welcome-file-list>
		<welcome-file>index.jsp</welcome-file>
	</welcome-file-list>

	<context-param>
		<param-name>filesPath</param-name>
		<param-value>C:\Users\Ashok\Desktop\Files\</param-value>
	</context-param>

</web-app>
```

=================================0=======================================

After Deploying the application, below screen will be displayed to upload the file into Server. We can change filesPath in web.xml file



Internationalization (I18N)

Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes. The term is frequently abbreviated as i18n (where 18 stands for the number of letters between the first i and last n in internationalization). Above definition essentially means making changes in your application such that it should be usable in multiple locales (or languages in simple words).

With the advent of globalization, the importance of internationalizing web applications has increased. This is important because web applications are accessed by users from various regions and countries. The Internet has no boundaries and neither should your web application. People all over the world access the Net to browse web pages that are written in different languages. For example, you can check your web-based email from virtually anywhere. A user in America can access the web and check her Facebook!. How does Facebook do it?

JAVA and Internationalization: Java Standard Edition, provides a rich set of API's to support Internationalization and Localization of an application. The internationalization API's provided with Java(SE) can easily adapt different messages, number, date, and currency formats according to different country and region conventions. The internationalization and Localization API's specified under Java SE platform is based on the Unicode 3.0 character encoding. Java provides two common classes to implement internationalization, **Locale and ResourceBundle**.

Internationalization and Localization: When you read about internationalizing applications, you will come across terms such as localization, character encoding, locales, resource bundles and so on.

Some of the commonly used terms
Internationalization or I18n is the process of enabling your application to cater to users from different countries and supporting different languages. With I18n, software is made portable between languages or regions. For example, the Yahoo! Web site supports users from English, Japanese and Korean speaking countries, to name a few. Localization or L10N on the other hand, is the process of customizing your application to support a specific location. When you customize your web application to a specific country say, Germany, you are localizing your application. Localization involves establishing on-line information to support a specific language or region. Though I18n and L10n appear to be at odds with each other, in reality they are closely linked.

Describing the Locale Class: The java.util.Locale class is used while creating internationalization Java's applications. The java.util.Locale class is a non-abstract final class packed into the java.util package. A Locale object encapsulates the language, country, and variant of the specific locale. These Locales affects language choice, collation, calendar usage, date and time formats, number and currency formats, and many other cultural sensitive data representations. All other locale-oriented classes use the locale object to adapt to the specific locale and provide the output accordingly.
The Locale class consists of three constructors:

Locale (String Language)
Locale (String language, String country)
Locale (String language, String country, String variant)

Parameters Of The locale Object:

There are three parameters which are ed in the Locale Objects:

Language, Country, Variant

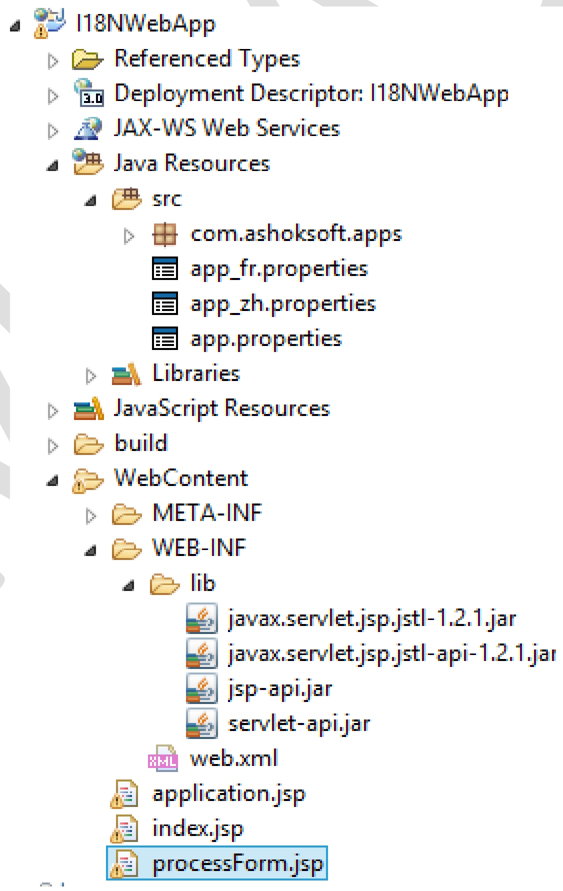
Describing The ResourceBundle Class: The ResourceBundles class provides the mechanism to separate user interface(ui) elements and other locale sensitive data from the application logic. ResourceBundle is an abstract base class that represents a container of resources.

It has two subclasses:
ListResourceBundle and PropertiesResourceBundle. When you are localizing your application, all the locale specific resources like text messages, icons and labels are stored in subclasses of the ResourceBundle. There will be one instance of the ResourceBundle per locale.The getBundle method in this class retrieves the appropriate ResourceBundle instance for a given locale. The location of the right bundle is implemented using an algorithm which is explained later in this section.

Internationalizing Web Applications: Internationalization is the process of preparing an application to support more than one language and data format. For internalizing java web application, we must first decide how to determine the user's language and locale preferences. Let's create an application named i18nWebx1, which demonstrate how to internationalize Web application:

This Example shows you Internationalization of language. This application gives the way of format for different languages. Here I have specified only three languages i.e.-English, Chinese, French. In the application given below we will develop a form for users from different countries. So user can select the from language as per suitability and operate further process.

Project Structure in Eclipse IDE



```
I18NWebApp
  Referenced Types
  Deployment Descriptor: I18NWebApp
  JAX-WS Web Services
  Java Resources
    src
      com.ashoksoft.apps
      app_fr.properties
      app_zh.properties
      app.properties
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
    WEB-INF
      lib
        javax.servlet.jsp.jstl-1.2.1.jar
        javax.servlet.jsp.jstl-api-1.2.1.jar
        jsp-api.jar
        servlet-api.jar
      web.xml
    application.jsp
    index.jsp
    processForm.jsp
```

Below are the Resource Bundles using in the project (properties files)

--app.properties--

```
newTitle=Formulaire de Demande
lastName=Pr\u00e9nom
firstName=Nom
postalCode=Code postal
password=Mot de pass\u00e9
submitForm=Valider
appInfo=L\u2019information de demadeur
```

--app_zh.properties--

```
newTitle=\u7533\u8acb\u8868\u683c
lastName=\u59d3
firstName=\u540d\u5b57
postalCode=\u90f5\u653f\u7de8\u865f
password=\u5bc6\u78bc
submitForm=\u63d0\u4ea4\u8868\u683c
appInfo=\u7533\u8acb\u4eba\u8cc7\u6599
```

--app_fr.properties--

```
newTitle=Formulaire de Demande
lastName=Pr\u00e9nom
firstName=Nom
postalCode=Code postal
password=Mot de pass\u00e9
submitForm=Valider
appInfo=L\u2019information de demadeur
```

---index.jsp--

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<body>
        <c:url value="application.jsp" var="engURL">
                <c:param name="locale" value="en_US" />
        </c:url>

        <a href="${engURL}"> English </a> |
        <c:url value="application.jsp" var="chineseURL">
                <c:param name="locale" value="zh_HK" />
        </c:url>

        <a href="${chineseURL}"> Chinese </a> |
        <c:url value="application.jsp" var="frenchURL">
                <c:param name="locale" value="fr_FR" />
        </c:url>

        <a href="${frenchURL}"> French </a>
</body>
</html>
```

--application.jsp--

```
<%@ page pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<html>
<c:set var="loc" value="en_US" />
<c:if test="${!(empty param.locale)}">
        <c:set var="loc" value="${param.locale}" />
</c:if>
<fmt:setLocale value="${loc}" />

<fmt:bundle basename="app">
        <head>
<title><fmt:message key="newTitle" /></title>
        </head>
        <body>
                <h1>
                        <fmt:message key="newTitle" />
                </h1>
                <c:url value="processForm.jsp" var="formActionURL" />
                <form action="${formActionURL}" method="post">
                        <table>
                                <tr>
                                        <td><fmt:message key="lastName" /></td>
                                        <td><input type="hidden" name="locale" value="${loc}" /> <input
                                                type="text" name="lastname" size="40" /></td>
                                </tr>
                                <tr>
                                        <td><fmt:message key="firstName" /></td>
                                        <td><input type="text" name="firstname" size="40" /></td>
                                </tr>
                                <tr>
                                        <td><fmt:message key="postalCode" /></td>
                                        <td><input type="text" name="postcode" size="40" /></td>
                                </tr>
                                <tr>
                                        <td><fmt:message key="password" /></td>
                                        <td><input type="password" name="pass" size="40" /></td>
                                </tr>
                                <tr>
                                        <td colspan="2" align="center"><input type="submit"
                                                value="<fmt:message key='submitForm'/>" /></td>
                                </tr>
                        </table>
                </form>
        </body>
                                <!—Including Languages page →
                                <jsp:include page="index.jsp" />
</fmt:bundle>
</html>
```

--processForm.jsp--

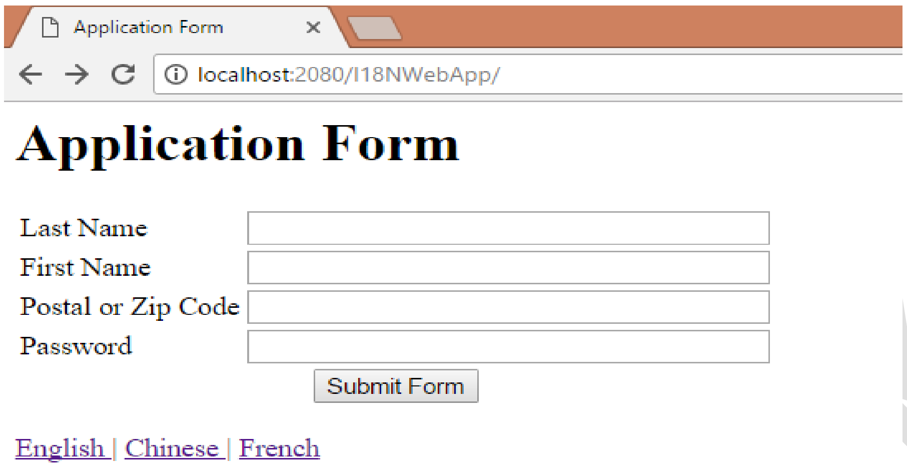
```
<%@ page pageEncoding="UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<fmt:requestEncoding value="UTF-8" />
<html>
<fmt:setLocale value="${param.locale}" />
<head>
<fmt:bundle basename="app">
        <title><fmt:message key="appInfo" /></title>
</head>
<body>
        <h1>
                <fmt:message key="appInfo" />
        </h1>
        <br>
        <table border="1">
                <tr>
                        <td><fmt:message key="lastName" /></td>
                        <td>${param.lastname}</td>
                </tr>
                <tr>
                        <td><fmt:message key="firstName" /></td>
                        <td>${param.firstname}</td>
                </tr>
                <tr>
                        <td><fmt:message key="postalCode" /></td>
                        <td>${param.postcode}</td>
                </tr>
                <tr>
                        <td><fmt:message key="password" /></td>
                        <td>${param.pass}</td>
                </tr>
        </table>
</body>
</fmt:bundle>
</html>
```

-----------------------------web.xml--------------------------------

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>I18NWebApp</display-name>
  <welcome-file-list>
    <welcome-file>application.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

--------------------------------------0---------------------------------

When User deployed this application, the first page would be like below

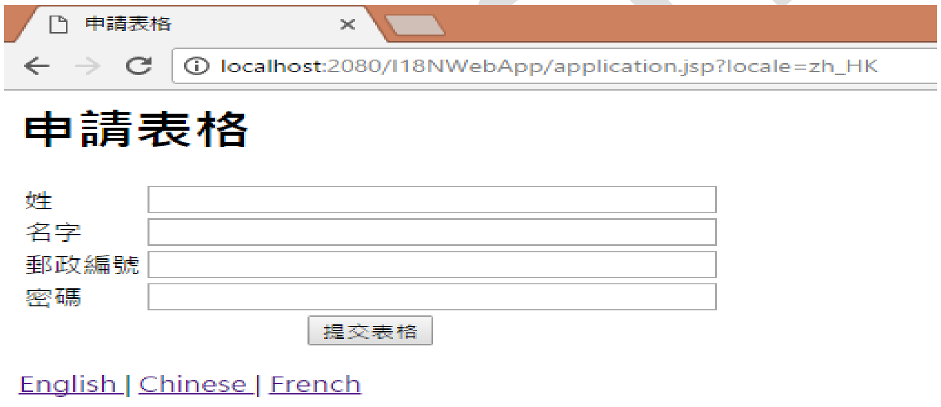


Application Form

Last Name	
First Name	
Postal or Zip Code	
Password	

English | Chinese | French

If User select language as 'Chinese' the page would be like below

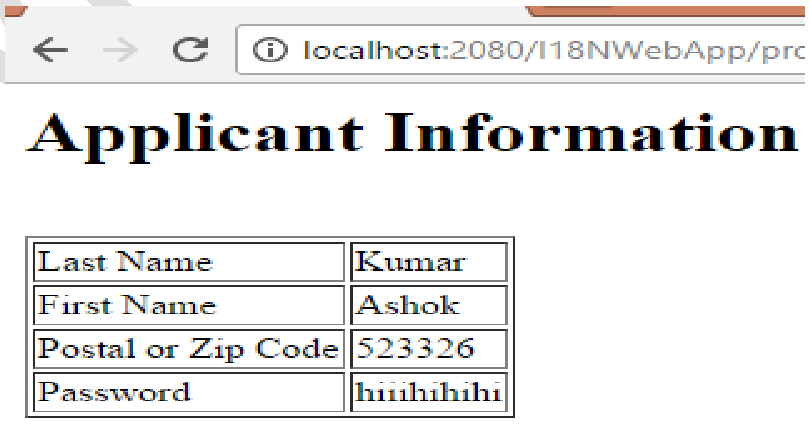


申請表格

姓	
名字	
郵政編號	
密碼	

English | Chinese | French

When User submit the form, form data will be displayed in selected language, like below



Applicant Information

Last Name	Kumar
First Name	Ashok
Postal or Zip Code	523326
Password	hiiihihihi

Design Patterns

A design pattern describes a proven solution to a recurring design problem, placing particular emphasis on the context and forces surrounding the problem, and the consequences and impact of the solution. There are many good reasons to use design patterns:

- ✓ They are proven: You tap the experience, knowledge and insights of developers who have used these patterns successfully in their own work.
- ✓ They are reusable: When a problem recurs, you don't have to invent a new solution; you follow the pattern and adapt it as necessary.
- ✓ They are expressive: Design patterns provide a common vocabulary of solutions, which you can use to express larger solutions succinctly.

It is important to remember, however, that design patterns do not guarantee success. You can only determine whether a pattern is applicable by carefully reading its description, and only after you've applied it in your own work can you determine whether it has helped.

Web is the very complex issues these days. Since the desire of the companies and organizations are increasing so the complexity and the performance of the web programming matters. Complexity with the different types of communication devices is increasing. The business is demanding applications using the web and many communication devices. So with the increase load of the data on the internet we have to take care of the architecture issue. To overcome these issues, we need to have idea about design models. There are two types of design models available in java, they are

1. **Model 1 Architecture**
2. **Model 2 (MVC) Architecture**

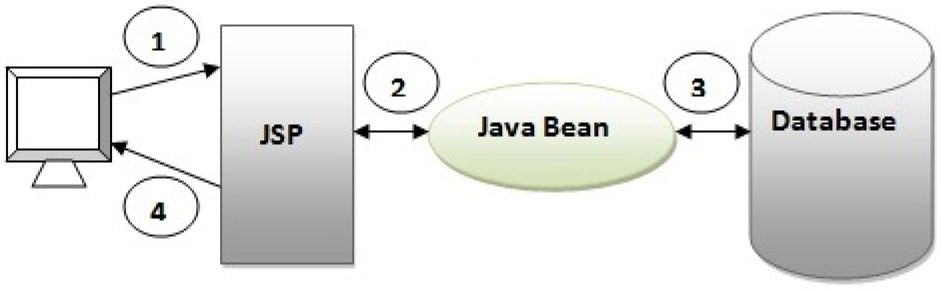
Model 1 Architecture

Servlet and JSP are the main technologies to develop the web applications.

Servlet was considered superior to CGI. Servlet technology doesn't create process, rather it creates thread to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area. Thus many subsequent requests can be easily handled by servlet.

Problem in Servlet technology Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

JSP overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated. You don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.



As you can see in the above figure, there is picture which show the flow of the model1 architecture.

Browser sends request for the JSP page
JSP accesses Java Bean and invokes business logic
Java Bean connects to the database and get/save data
Response is sent to the browser which is generated by JSP
Advantage of Model 1 Architecture

- ✓ Easy and Quick to develop web application

Disadvantage of Model 1 Architecture

- ➢ Navigation control is decentralized since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
- ➢ Time consuming You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.
- ➢ Hard to extend It is better for small applications but not for large applications.

Model 2 Architecture

Model View Controller (MVC) is a software architecture pattern, commonly used to implement user interfaces: it is therefore a popular choice for architecting web apps. In general, it separates out the application logic into three separate parts, promoting modularity and ease of collaboration and reuse. It also makes applications more flexible and welcoming to iterations.

The MVC pattern was first described in 1979 by Trygve Reenskaug, then working on Smalltalk at Xerox research labs. The original implementation is described in depth in the influential paper “Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller”.

Smalltalk’s MVC implementation inspired many other GUI frameworks such as:

- ➢ The NeXTSTEP and OPENSTEP development environments encourage the use of MVC. Cocoa and GNUstep, based on these technologies, also use MVC.
- ➢ Microsoft Foundation Classes (MFC) (also called Document/View architecture)
- ➢ Java Swing
- ➢ The Qt Toolkit (since Qt4 Release).
- ➢ XForms has a clear separation of model (stored inside the HTML head section) from the presentation (stored in the HTML body section). XForms uses simple bind commands to link the presentation to the model.

MVC stands for Model, View and Controller. MVC separates application into three components - Model, View and Controller.

Model: The Model is where data from the controller and sometimes the view is actually passed into, out of, and manipulated. Keeping in mind our last example of logging into your web-based email, the model will take the username and password given to it from the controller, check that data against the stored information in the database, and then render the view accordingly. For example, if you enter in an incorrect password, the model will tell the controller that it was incorrect, and the controller will tell the view to display an error message saying something to the effect of "Your username or password is incorrect."

Model is a data and business logic.

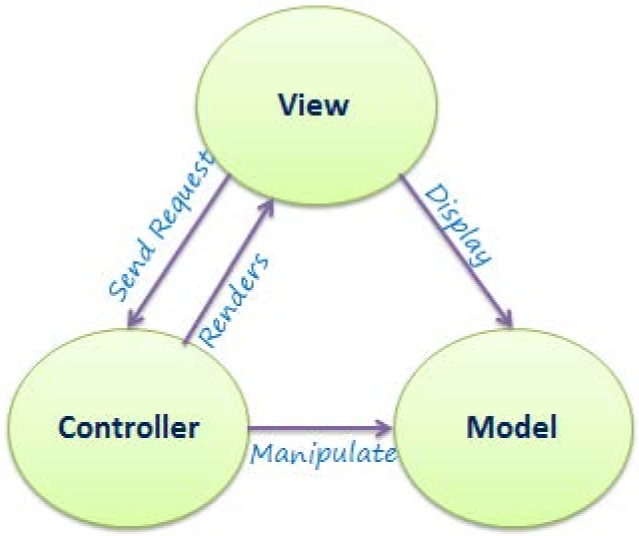
View: In a web-based application, **the view** is exactly what it sounds like: the visible interface that the user interacts with, displaying buttons, forms, and information. Generally speaking, the controller calls up the view after interacting with the model, which is what gathers the information to display in the particular view.

View is a User Interface.

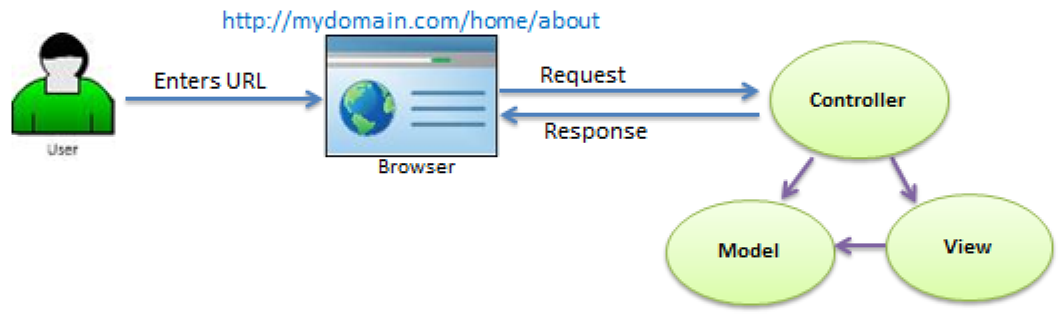
Controller: The Controller is essentially the traffic cop of the application, directing traffic to where it needs to go, figuring out which view it needs to load up, and interacting with the appropriate models. For example, when you go to login to your email on a website, the controller is going to tell the application that it needs to load the login form view. Upon attempting to login, the controller will load the model that handles logins, which will check if the username and password match what exists within the system. If successful, the controller will then pass you off to the first page you enter when logging in, such as your inbox. Once there, the inbox controller will further handle that request.

Controller is a request handler.

The following figure illustrates the interaction between Model, View and Controller



The following figure illustrates the flow of the user's request



As per the above figure, when the user enters a URL in the browser, it goes to the server and calls appropriate controller. Then, the Controller uses the appropriate View and Model and creates the response and sends it back to the user.

Points to Remember:

- MVC stands for Model, View and Controller.
- Model is responsible for maintaining application data and business logic.
- View is a user interface of the application, which displays the data.
- Controller handles user's requests and renders appropriate View with Model data.

Though MVC comes in different flavours, the control flow generally works as follows:

- The user interacts with the user interface in some way (e.g., user presses a button)
- A controller handles the input event from the user interface, often via a registered handler or callback.
- The controller accesses the model, possibly updating it in a way appropriate to the user's action (e.g., controller updates user's shopping cart). Complex controllers are often structured using the command pattern to encapsulate actions and simplify extension.
- A view uses the model to generate an appropriate user interface (e.g., view produces a screen listing the shopping cart contents). The view gets its own data from the model. The model has no direct knowledge of the view. (However, the observer pattern can be used to allow the model to indirectly notify interested parties, potentially including views, of a change.)
- The user interface waits for further user interactions, which begins the cycle anew.

Conclusion

Now that you know the basic concepts of the MVC pattern, you're probably wondering what makes it so special? Essentially, it allows for the programmer to isolate these very separate pieces of code into their own domain, which makes code maintenance and debugging much simpler than if all of these items were chunked into one massive piece. If I have a problem with an application not displaying an error message when it should, I have a very specific set of locations to look to see why this is not happening. First I would look at the "Login Controller" to see if it is telling the view to display the error. If that's fine, I would look at the "Login Model" to see if it is passing the data back to the controller to tell it that it needs to show an error. Then if that's correct, the last place it could be happening would be in the "Login View."

Using this development pattern allows for very easy maintenance, as well as independent development of pieces of the same system by different programmers, which makes for quick turnover of applications all while still maintaining a very high standard of quality for the application.

Servlet Interview Questions

1. What is different between web server and application server?
2. Which HTTP method is non-idempotent?
3. What is the difference between GET and POST method?
4. What is MIME Type?
5. What is a web application and what is it’s directory structure?
6. What is a servlet?
7. What are the advantages of Servlet over CGI?
8. What are common tasks performed by Servlet Container?
9. What is ServletConfig object?
10. What is ServletContext object?
11. What is difference between ServletConfig and ServletContext?
12. What is Request Dispatcher?
13. What is difference between PrintWriter and ServletOutputStream?
14. Can we get PrintWriter and ServletOutputStream both in a servlet?
15. How can we create deadlock situation in servlet?
16. What is the use of servlet wrapper classes?
17. What is SingleThreadModel interface?
18. Do we need to override service() method?
19. Is it good idea to create servlet constructor?
20. What is difference between GenericServlet and HttpServlet?
21. What is the inter-servlet communication?
22. Are Servlets Thread Safe? How to achieve thread safety in servlets?
23. What is servlet attributes and their scope?
24. How do we call one servlet from another servlet?
25. How can we invoke another servlet in a different application?
26. What is difference between ServletResponse sendRedirect() and RequestDispatcher forward() method?
27. Why HttpServlet class is declared abstract?
28. What are the phases of servlet life cycle?
29. What are life cycle methods of a servlet?
30. Why we should override only no-agrs init() method.
31. What is URL Encoding?
32. What are different methods of session management in servlets?
33. What is URL Rewriting?
34. How does Cookies work in Servlets?
35. How to notify an object in session when session is invalidated or timed-out?
36. What is the difference between encodeRedirectUrl and encodeURL?
37. Why do we have servlet filters?
38. What is the effective way to make sure all the servlets are accessible only when user has a valid session?
39. Why do we have servlet listeners?
40. How to handle exceptions thrown by application with another servlet?
41. What is a deployment descriptor?
42. How to make sure a servlet is loaded at the application startup?
43. How to get the actual path of servlet in server?

44. How to get the server information in a servlet?
45. Write a servlet to upload file on server.
46. How do we go with database connection and log4j integration in servlet?
47. How to get the IP address of client in servlet?
48. What are important features of Servlet 3?
49. What are different ways for servlet authentication?
50. How can we achieve transport layer security for our web application?

JSP Interview Questions

1. What is JSP and why do we need it?
2. What are the JSP lifecycle phases?
3. What are JSP lifecycle methods?
4. Which JSP lifecycle methods can be overridden?
5. How can we avoid direct access of JSP pages from client browser?
6. What are different types of comments in JSP?
7. What is Scriptlet, Expression and Declaration in JSP?
8. What are JSP implicit objects?
9. Can we use JSP implicit objects in a method defined in JSP Declaration?
10. Which implicit object is not available in normal JSP pages?
11. What are the benefits of PageContext implicit object?
12. How do we configure init params for JSP?
13. Why use of scripting elements in JSP is discouraged?
14. Can we define a class in a JSP Page?
15. How can we disable java code or scripting in JSP page?
16. Explain JSP Action Elements or Action Tags?
17. What is difference between include directive and jsp:include action?
18. What is JSP Expression Language and what are it's benefits?
19. What are JSP EL implicit objects and how it's different from JSP implicit Objects?
20. How to use JSP EL to get HTTP method name?
21. What is JSP Standard Tag Library, provide some example usage?
22. What are the types of JSTL tags?
23. What is JSP Custom Tag and what are it's components?
24. Give an example where you need JSP Custom Tag?
25. Why don't we need to configure JSP standard tags in web.xml?
26. How can we handle exceptions thrown by JSP service method?
27. How do we catch exception and process it using JSTL?
28. How do we print "<br> creates a new line in HTML" in JSP?
29. What is jsp-config in deployment descriptor?
30. How to ignore the EL expression evaluation in a JSP?
31. When will Container initialize multiple JSP/Servlet Objects?
32. Can we use JavaScript with JSP Pages?
33. How can we prevent implicit session creation in JSP?
34. What is difference between JspWriter and Servlet PrintWriter?
35. How can we extend JSP technology?
36. Provide some JSP Best Practices?

========Happy Learning=========